# Projeto TPG

# Nome: Felipe Augusto Pedroso

## Introdução

Para o projeto proposto na aula de TPG foi escolhido o desenvolvimento de um jogo genérico de cartas, aonde os algoritmos desenvolvidos serão:

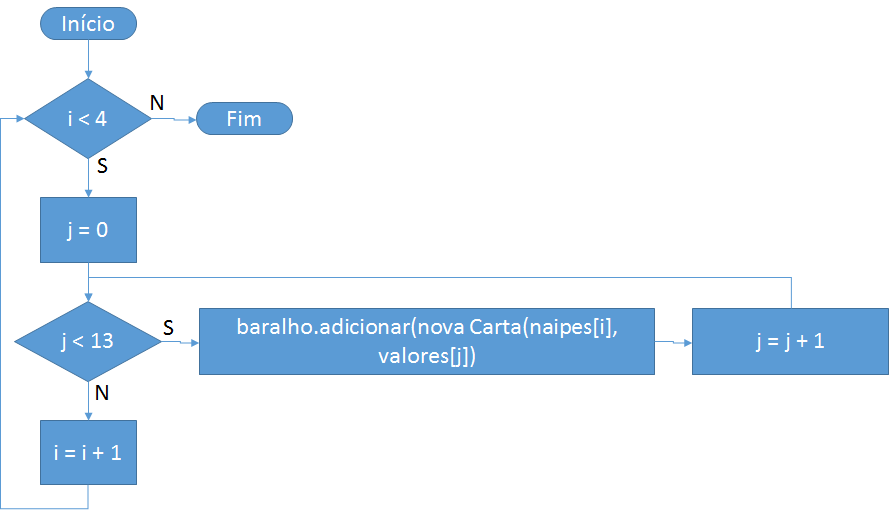
* *CriarBaralho*: que tem como propósito criar um baralho dada a combinação de naipes (Copas, Espadas, Ouros e Paus) e valores (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K) num total de 52 cartas
* *Embaralhar*: aonde um baralho criado pelo algoritmo “CriarBaralho” possa ser embaralhado, fazendo com que suas cartas estejam ordenadas randomicamente.
* *DistribuirCartas*: após a criação e embaralhamento das cartas, este algoritmo irá distribuir uma quantidade n de cartas para uma dada quantidade de jogadores

As implementações dos algoritmos em Java estão disponíveis em anexo ao trabalho e também no repositório de código: <https://github.com/felipepedroso/Projeto-TPG>

## Parte 1 – Algoritmos

### Algoritmo 1 - “CriarBaralho”

* **Fluxograma**



* **Pseudocódigo**

inicio

enumeracao Naipes

enumeracao Valores

var cartas : Lista<Carta>

var carta : Carta

var naipe : Naipes

var valor : Valores

para cada naipe de Naipes

para cada valor de Valores

carta = nova Carta (naipe, valor)

cartas.adicionarCarta(carta)

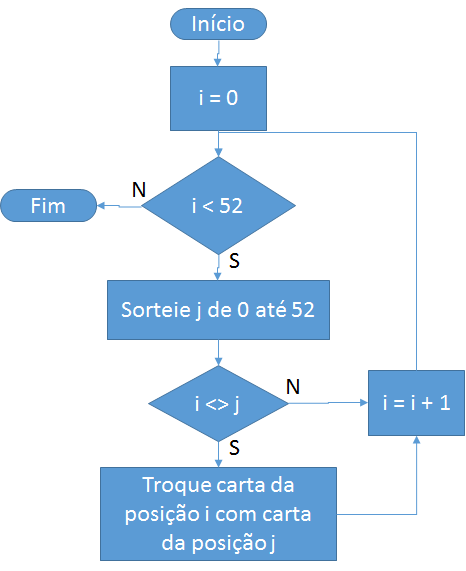
fim para

fim para

fim

### Algoritmo 2 – “Embaralhar”

* **Fluxograma**



* **Pseudocódigo**

inicio

var cartas : Lista<Cartas>

var aux : Carta

var tamanhoBaralho, i, j : inteiro

tamanhoBaralho <- cartas.tamanho()

para i de 0 ate (tamanhoBaralho - 1) passo 1

j <- sorteia (0, tamanhoBaralho - 1)

se i <> j

aux = carta[i]

carta[i] = carta[j]

carta[j] = aux

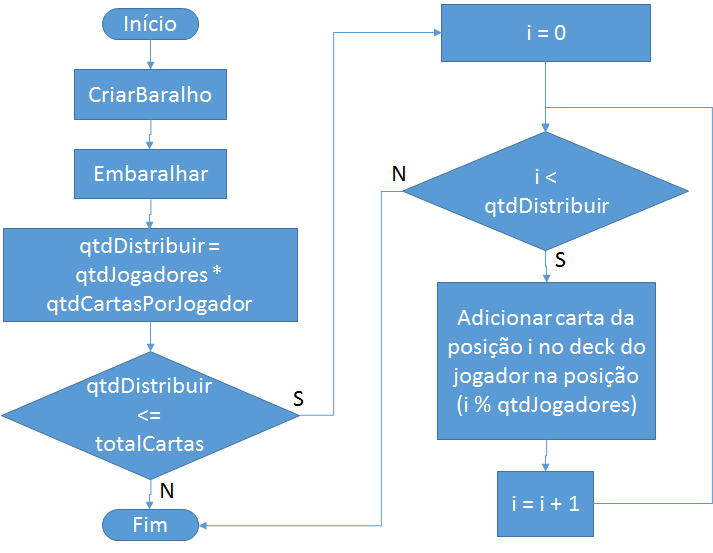
fim se

fim para

fim

### Algoritmo 3 – “DistribuirCartas”

* **Fluxograma**



* **Pseudocódigo**

inicio

constante inteiro qtdJogadores <- 2

constante inteiro qtdCartasPorJogador <- 3

var cartas : Lista<Carta>

var jogadores : Lista<Deck>

var carta : Carta

cartas <- CriarBaralho()

Embaralhar(cartas)

para i de 0 ate qtdJogadores \* qtdCartasPorJogador passo 1

carta <- cartas[i]

deckJogadores[i % qtdJogadores].adicionarCarta(carta)

fim para

fim

## Parte 2 – Extreme Programming

### Definição do Time

* **Project Owner:**
  + Mr. Church
* **Scrum Master**
  + Barney Ross
  + Lee Christmas (backup)
* **Time de Desenvolvimento**
  + Gunner Jensen
  + Yin Yang
  + Hale Caesar
  + Lee Christmas

### Planejamento

O desenvolvimento do produto se dará em 3 Sprints com 2 semanas em cada um. Cada Sprint irá contemplar o desenvolvimento da funcionalidade, documentação e desenvolvimento dos testes. Dada a importância dos algoritmos selecionados, foi determinado que o desenvolvimento do produto será na seguinte ordem:

* Sprint 1: Algoritmo de criação do baralho (incluindo a modelagem dos objetos envolvidos)
* Sprint 2: Algoritmo de embaralhamento
* Sprint 3 Algoritmo de distribuição de cartas

### Testes dos três algoritmos

Os testes dos algoritmos estão inclusos no código em anexo.

## Anexo – Código Fonte (Java)

* **CriarBaralho**

public List<Card> **criarBaralho**() {

Suit[] naipes = Suit.*values*();

Rank[] valores = Rank.*values*();

List<Card> baralho = new ArrayList<>();

for (int i = 0; i < naipes.length; i++) {

for (int j = 0; j < valores.length; j++) {

Card carta = new Card(valores[j], naipes[i]);

baralho.add(carta);

}

}

return baralho;

}

* **Embaralhar**

public void **embaralhar**(List<Card> cartas) {

Random random = new Random(System.*nanoTime*());

for (int i = 0; i < cartas.size(); i++) {

int j = random.nextInt(cartas.size());

if (i != j) {

Card aux = cartas.get(i);

cartas.set(i, cartas.get(j));

cartas.set(j, aux);

}

}

}

* **DistribuirCartas**

public List<Deck> **distribuirCartas**(List<Card> baralho, int quantidadeJogadores, int quantidadeCartasPorJogador) {

List<Deck> deckJogadores = new ArrayList<>();

int quantidadeCartas = quantidadeJogadores \* quantidadeCartasPorJogador;

if (quantidadeCartas <= baralho.size()) {

for (int i = 0; i < quantidadeJogadores; i++) {

deckJogadores.add(new Deck(String.*format*("Jogador %d", i)));

}

for (int i = 0; i < quantidadeCartas; i++) {

Card carta = baralho.remove(i);

deckJogadores.get(i % quantidadeJogadores).adicionarCarta(carta);

}

}

return deckJogadores;

}

* **Testes**

JogoCartas jogo = new JogoCartas();

// Testando a criação do baralho

List<Card> baralho1 = jogo.criarBaralho();

final int quantidadeMaximaCartas = Rank.*values*().length \* Suit.*values*().length;

System.*out*.println("-----------------------------------");

System.*out*.println("Teste do algoritmo **\"**CriarBaralho**\"**");

System.*out*.println(String.*format*("Testando se a lista de cartas não retornou nula: %b", baralho1 != null));

if (baralho1 != null && !baralho1.isEmpty()) {

System.*out*.println(String.*format*("Baralho criado: %s", baralho1.toString()));

System.*out*.println(String.*format*("Testando se todas as combinações de cartas foram criadas: %b", baralho1.size() == quantidadeMaximaCartas));

System.*out*.println(String.*format*("Testando se o baralho sempre é criado da mesma maneira: %b", baralho1.containsAll(jogo.criarBaralho())));

boolean duplicacasCriarBaralho = false;

for (Card carta : baralho1) {

//System.out.println(String.format("Frequency of %s inside the deck: %d", carta.toString(), Collections.frequency(baralho1, carta)));

if (Collections.*frequency*(baralho1, carta) > 1) {

duplicacasCriarBaralho = true;

break;

}

}

System.*out*.println(String.*format*("Testanto se existem cartas duplicadas: %b", !duplicacasCriarBaralho));

} else {

return;

}

// Testando o embaralhamento

List<Card> copiaBaralho = Collections.*EMPTY\_LIST*;

Collections.*copy*(baralho1, copiaBaralho);

System.*out*.println("-----------------------------------");

System.*out*.println("Teste do algoritmo **\"**Embaralhar**\"**");

jogo.embaralhar(baralho1);

System.*out*.println(String.*format*("Baralho após embaralhamento: %s", baralho1.toString()));

System.*out*.println(String.*format*("Testando se nenhuma carta foi removida: %b", baralho1.size() == quantidadeMaximaCartas));

System.*out*.println(String.*format*("Testando se todas as cartas estão no baralho original após embaralhar: %b", baralho1.containsAll(copiaBaralho)));

boolean testeDuplicadasEmbaralhar = false;

for (Card carta : baralho1) {

//System.out.println(String.format("Frequency of %s inside the deck: %d", carta.toString(), Collections.frequency(baralho1, carta)));

if (Collections.*frequency*(baralho1, carta) > 1) {

testeDuplicadasEmbaralhar = true;

break;

}

}

System.*out*.println(String.*format*("Testanto se existem cartas duplicadas após embaralhar: %b", !testeDuplicadasEmbaralhar));

// Testando a distribuição de cartas

System.*out*.println("-----------------------------------");

System.*out*.println("Teste do algoritmo **\"**DistribuirCartas**\"**");

final int quantidadeJogadores = 5;

final int quantidadeCartasPorJogador = 4;

List<Deck> deckJogadores = jogo.distribuirCartas(baralho1, quantidadeJogadores, quantidadeCartasPorJogador);

System.*out*.println(String.*format*("Testando se o método distribuirCartas não retorna null: %b", deckJogadores != null));

if (deckJogadores != null && !deckJogadores.isEmpty()) {

System.*out*.println(String.*format*("Testando se foram gerados decks para todos os jogadores: %b", deckJogadores.size() == quantidadeJogadores));

System.*out*.println(String.*format*("Testando se a quantidade de cartas distribuidas foram removidas do baralho: %b", baralho1.size() == quantidadeMaximaCartas - (quantidadeJogadores \* quantidadeCartasPorJogador)));

System.*out*.println("Testando a quantidade de cartas na mão para cada jogador:");

for (Deck deck : deckJogadores) {

System.*out*.println(String.*format*("**\t**> Testando se o jogador **\"**%s**\"** tem a quantidade de cartas adequada no deck: %b", deck.getNomeJogador(), deck.getCartas().size() == quantidadeCartasPorJogador));

System.*out*.println(String.*format*("**\t**> Testando se as cartas do jogador **\"**%s**\"** foram removidas do baralho: ", deck.getNomeJogador()));

for (Card carta : deck.getCartas()) {

System.*out*.println(String.*format*("**\t\t**- Testando se a carta %s foi removida do baralho: %b", carta.toString(), !baralho1.contains(carta)));

}

}

List<Card> cartasDistribuidas = new ArrayList<>();

for (int i = 0; i < deckJogadores.size(); i++) {

cartasDistribuidas.addAll(deckJogadores.get(i).getCartas());

}

boolean duplicadasDistribuir = false;

for(Card carta : cartasDistribuidas){

if (Collections.*frequency*(cartasDistribuidas, carta) > 1) {

duplicadasDistribuir = true;

break;

}

}

System.*out*.println(String.*format*("Testando se existem cartas duplicadas entre as cartas distribuidas: %b", !duplicadasDistribuir));

} else {

return;

}

System.*out*.println(String.*format*("Testando se o metodo retorna um array vazio quando a quantidade de cartas é insuficiente: %b", jogo.distribuirCartas(baralho1, 10, 10).isEmpty()));