UFMG/ICEx/DCC - Programação modular
Trabalho prático 2 - *Cartas na mesa(Truco)*

Pedro Paulo Valadares Brum
Rafael Grandsire de Oliveira

16 de maio de 2017

1 Introdução

Neste trabalho prático tivemos que implementar um jogo de cartas sequindo os princípios da Orientação a Objetos e utilizando a linguagem Java. Além disso, deveríamos tentar aplicar de forma correta os padrões de projeto aplicáveis a esse contexto. O jogo implementado foi o Truco. Para jogar truco são necessárias duas duplas de pessoas, sendo que cada pessoa começa com 3 cartas.

Como o objetivo principal do trabalho é aplicar os princípios da Orientação a Objetos e utilizar os padrões de projeto aplicáveis, as regras do jogo não foram seguidas à risca e, portanto, foi implementada um versão mais simples do jogo. Além disso, foi criada um interface gráfica via *Swing* para os jogadores.

Como o jogo deveria ser implementado seguindo-se os princípios da Orientação a Objeto, espera-se que o programa tenha uma boa abstração do problema, que utilize de forma inteligente o encapsulamento, que use corretamente o princípio da herança e que aplique o polimorfismo de forma a melhorar o código. Espera-se também um programa bem modularizado, com a separação em conjuntos de módulos, classes com independência de funcionamento, separação de resposabilidades, entre outras características. Além disso, espera-se também o uso do princípio das mensagens que propõe a comunicação entre objetos, envio e recebimento de mensagens e utilização de contrado firmado entre as partes.

2 Regras

O jogo funciona, basicamente, da seguinte forma: no início do jogo cada time começa com 0 pontos e a cada rodada, cada jogador inicia com 3 cartas. Quando um time guanha uma rodada em que não foi pedido truco ele ganha 1 ponto, se o time ganha um rodada em que foi pedido truco ele ganha 3 pontos. Um time sempre pode pedir truco, a menos

que o outro time já tenha pedido. Se um time pede truco o outro time pode aceitar ou desistir. O jogo termina quando um dos times possui 12 ou mais pontos, ou seja, quando um time vence o jogo.

3 Implementação

3.1 Truco

Este módulo é o módulo principal e representa o jogo Truco. Nele instanciamos objetos das classes *Time*, *Jogador*, *Baralho*, *Exibivel* e *Alertador*. Como foi utilizada interface gráfica o própio usuário escolhe as cartas a cada rodada. De maneira resumida, o código embaralha o baralho, distribui as cartas entre os jogadores e, com base nas cartas jogadas pelos jogadores, descobre quem venceu a rodada. Além disso, a cada rodada os times tem a opção de pedir truco e, dessa forma, a rodada passa a valer mais pontos caso um dos times aceite o truco.

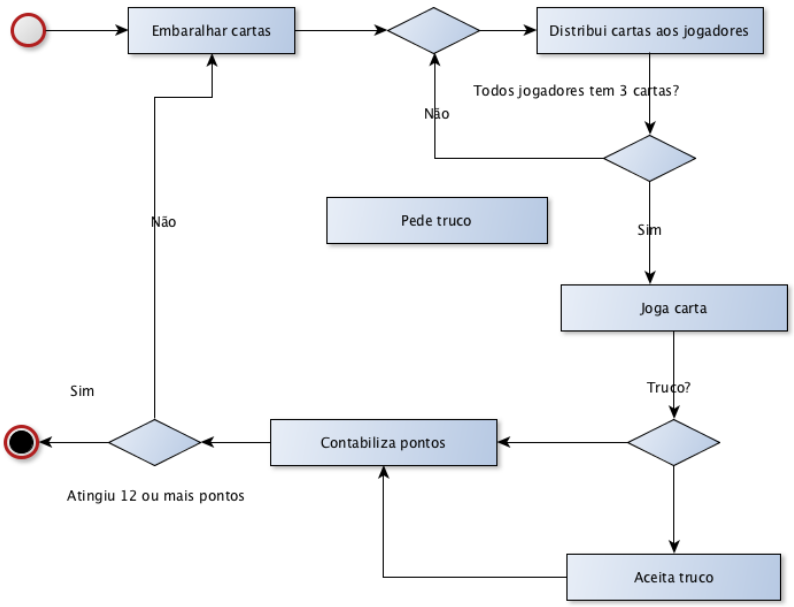

Figura 1: Diagrama de atividades

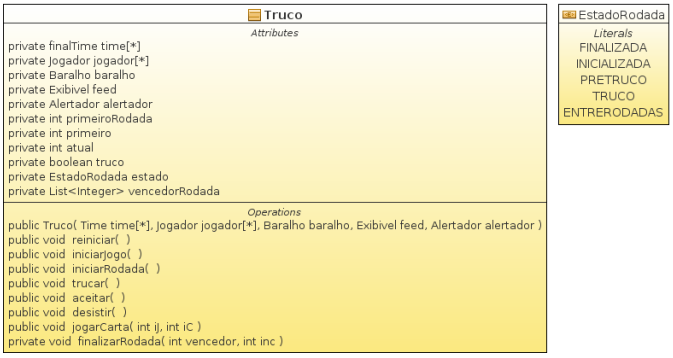

Figura 2: Truco

Atributos:

time - Lista com os times do jogo - *private final*.

jogador - Lista com os jogadores - *private final*.

baralho - Conjunto de cartas - *private final final*.

feed - - *private final*.

alertador - - *private final*.

primeiroRodada - primeiro jogador a da rodada - *private*.

primeiro - primeiro jogador a jogar carta - *private*.

atual - jogador que joga carta na rodada - *private*.

truco - indica pedido de truco por um dos times - *private*.

estado - indica o estado da jogada(Finalizada, inicializada, pretruco, truco, entrerodadas) - *private*.

vencedorRodada - armazenas os times vencedores de cada rodada do jogo - *private*.

Os atributos da classe são declarados como *private*, e dessa forma, só podem ser acessados por métodos da mesma classe.

Métodos:

Truco - Construtor da classe - *public*.

reiniciar - reinicia jogo - finaliza rodada corrente - *private static*.

iniciarJogo - inicia o jogo - - *public*.

iniciarRodada - *public* - inicia uma rodada do jogo: se o a rodada estiver no estado

FINALIZADA, embaralha-se o baralho, distribui-se as cartas aos jogadores, a rodada atual passa a ser a primeira rodada e um novo jogo se inicia.

trucar - um dos times pede truco: estado da rodada passa a ser igual a PRETRUCO - *public*.

aceitar - um dos times aceita o truco: estado da rodada passa ser igual a TRUCO - *public*.

desistir - um dos times desiste do jogo após o outro time pedir truco: o time pode desistir no estado PRETRUCO ou no estado ENTRERODADAS - *public*.

jogarCarta - um dos jogadores joda a carta indicada - *public*.

finalizarRodada - finalizada rodada atual: o número de ponto do time vencedor é incrementado, o estado da rodada passa a ser FINALIZADA. Se o número de pontos do time vencedor da rodada for maior ou igual a 12 o jogo termina. - *private*.

Os métodos foram declarados como *public* e , portanto, podem ser acessados de qualquer outra parte do código. Os métodos *formatInstrucao* e *isDump* foram declarados também como *static* e, portanto, são acessados diretamente pela classe, são resolvidos em tempo de compilação, não podem ser sobrescritos e só podem acessar atributos e métodos *static*.

3.2 Carta

Neste módulo foi implementado a classe *Carta* que representa as cartas do baralho. Ela foi declarada como public, pois ela pode precisar ser utilizada em qualquer lugar do programa. Essa classe possui os atributos *simbolo*, *valor* e *cor*.

Atributos:

simbolo - símbolo da carta - *private final*.

valor - valor da carta no jogo - *private final*.

cor - cor da carta - *private final*.

Os atributos da classe são declarados como *private final*, e dessa forma, só podem ser utilizadas detro da própria classe. Além disso, elas são utilizadas como constanstes, ou seja, para cada objeto instanciado, elas possuem sempre o mesmo valor.

Métodos:

Carta - Construtor da classe - *public*.

getSimbolo - permite o acesso ao atributo simbolo - *public*.

getValor - permite o acesso ao atributo valor - *public*.

getCor - permite o acesso ao atributo cor - *public*.

compareTo - retorna o resultado da comparação entre duas cartas - *public*.

toString - retorna o símbolo da carta - *public*.

Exibivel - interface: metódos que devem ser implementados para o funcionamento da interação com o usuário - *public interface*.

Os métodos foram declarados como *public*, pois podem ser utilizados em qualquer local do programa.

3.3 Baralho

Neste módulo foi implementado a classe *Baralho* que representa o baralho do jogo. Ela foi declarada como public, pois ela pode precisar ser utilizada em qualquer lugar do programa. Essa classe possui os atributos *instance*, *baralho* e *topo*. Nessa classe, o padrão de projeto Singleton foi utilizado e será abordado com mais detalhes posteriormente.

Atributos:

instance - - *private static*.

baralho - lista de cartas - *private final*.

topo - indica o topo do baralho - *private*.

Os atributos da classe são declarados como *private*, e dessa forma, só podem ser utilizadas detro da própria classe. O atributo *baralho* foi declarado como *private final* e, portanto, é utilizado como constanste, ou seja, para cada objeto instanciado, possui sempre o mesmo valor.

Métodos:

Baralho - Construtor da classe: adiciona as cartas no baralho - *private*.

getTopo - permite o acesso ao atributo topo que representa a carta do topo do baralho - *public*.

getInstance - se o não existir baralho, um novo baralho é criado. - *public static*.

embaralhar - embaralha o baralho - *public*.

Os métodos foram declarados como *public*, pois podem ser utilizados em qualquer local do programa.

3.4 Jogador

Neste módulo foi implementado a classe *Jogador* que representa os jogadores. Ela foi declarada como public, pois ela pode precisar ser utilizada em qualquer lugar do programa. Essa classe possui os atributos *carta*, *mesa*, *exibivelCarta*, *exibivelJogada*.

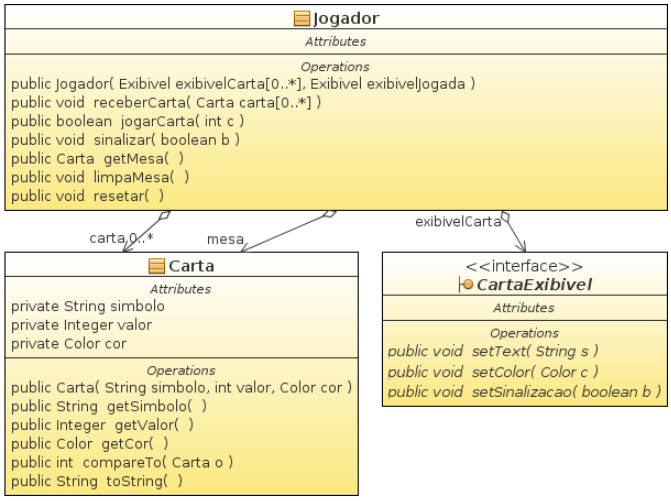

Figura 3: Jogador

Atributos:

carta - Lista de cartas que o jogador possui - *private*.

mesa - Carta que jogador escolheu na rodada - *private*.

exibivelCarta - cartas do jogador - *private final*.

exibivelJogada - carta que o jogar colocou na mesa - *private final*.

Os atributos da classe são declarados como *private*, e dessa forma, só podem ser utilizadas detro da própria classe. Os atributos *exibicelCarta* e *exibivelJogada* foram declaradas como *private final* e, portanto, são utilizadas como constanstes, ou seja, para cada objeto instanciado, possuem sempre o mesmo valor.

Métodos:

Jogador - Construtor da classe - *public*.

receberCarta - Distribui cartas ao jogador: cada um dos 4 jogadores recebe 3 cartas do baralho - *public*.

jogarCarta - coloca um carta do jogador na mesa - *public*.

sinalizar - sinaliza carta - *public*.

getMesa - Construtor da classe - *public*.

limpaMesa - retira a carta do jogador da mesa - *public*.

resetar - retira as cartas da "mão"do jogador - *public*.

Os métodos foram declarados como *public*, pois podem ser utilizados em qualquer local do programa.

3.5 Time

Neste módulo foi implementado a classe *Time* que representa uma dupla do jogo Truco. Ela foi declarada como public, pois ela pode precisar ser utilizada em qualquer lugar do programa. Essa classe possui os atributos: *pontos* e *exibivelPontos*.

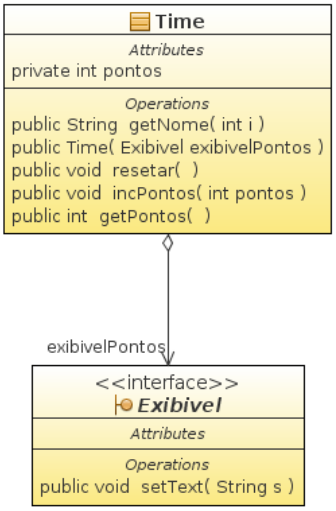

Figura 4: Time

Atributos:

pontos - número de pontos que a dupla possui no jogo - *private*.

exibivelPontos - número exibido na interface - *private final*.

Os atributos da classe são declarados como *private*, e dessa forma, só podem ser utilizadas detro da própria classe. O atributo *exibivelPontos* foi declarada como *private final* e, portanto, é utilizado como constanste, ou seja, para cada objeto instanciado, o atributo possui sempre o mesmo valor.

Métodos:

Time - Construtor da classe - *public*.

getNome - pemite acesso ao nome do time: horizontal/vertical - *public*.

resetar - reseta a quantidade de pontos do time - *public*.

incPontos - incrementa uma quantidade de pontos no número de pontos do time - *public*.

Os métodos foram declarados como *public*, pois podem ser utilizados em qualquer local do programa.

3.6 Decisões de Manutenabilidade

Toda a interface de interação do usuário com o jogo foi pensada de maneira genérica o suficiente que, para que uma nova versão do jogo seja implementada, basta que os objetos que representarão e realização as interações implementem as interfaces definidas. Nesse caso, foram criadas três classes diferentes, que extendiam JButton, para as cartas, JLabel, para os times e mensagens, e JOptionPane, para emitir mensagens de alertas sobre o estado do jogo.

3.7 Padrões de projeto utilizados

Nesse trabalho aplicou-se os sequintes padrões de projeto: *Singleton*, *Observer*, *State*, *Visitor* e *Adapter*.

• Singleton: A classe *Baralho* possui apenas uma instância durante toda execução. Dentro da classe Baralho, pode-se observar a presença de um atributo *instance* e do contrutor da classe Baralho com o modificador *private*. Esse padrão de projeto garante que a classe tem apenas uma instância, e prover um ponto de acesso global a ela. Essa instância pode ser estendida sem modificar o código, utilizando-se o princípio da herança. Como consequência temos: acesso controlado a uma única instância, espaço de nomes reduzido e refinamento de operações e de representação.

• Observer: Define uma dependência entre objetos, que são notificados de uma mudança de estado. A mudança de estado de um objeto de classe *Time* requer que outros objetos sejam notificados e modificados. Isso ocorre por exemplo com os objetos das classes utilizadas para criação da interface gráfica. Quando um time ganha uma rodada, necessáriamente o número de pontos que aparece na interface gráfica também aumenta.

• State: Permite que o objeto altere o comportamento quando seu estado interno se modifica. A mudança de estado de um objeto da classe *Truco* altera o seu próprio comportamento. Existem 5 estados internos possíveis: FINALIZADA, INICIALIZADA, PRETRUCO, TRUCO e ENTRERODADAS. O comportamento desse objeto, portanto, depende de um estado que apenas é conhecido em tempo de execução. Assim, operações possuem número alto de condicionais. Como consequências temos: separaão dos comportamentos e transição de estados explícitas.

• Visitor: Permite que uma operação seja executada sobre um conjunto de objetos em tempo de execução. A cada rodada e cada jogada de um jogador operações são realizadas sobre os conjuntos de objetos durante o jogo, ou seja, em tempo de execução do código.

• Adapter: Para garantir que o programa funcione independente da interface de interação, vários componentes da Interface Gráfica houveram de ser Extendidos e Adaptados. As classes JButtonCarta, JLabelExibivel e JOptionPaneAlertador extendem elementos originais da Swing e implementa interfaces de modo a adaptar a exibição do fluxo do jogo para a interface gráfica.

3.8 Interface Gráfica - Operando Sobre o Truco

A class que opera sobre o Truco é a GUITruco. Além de chamar as operações fornecidas por Truco, ela possui objetos que implementam as interfaces de exibição e interação do usuário com o jogo.

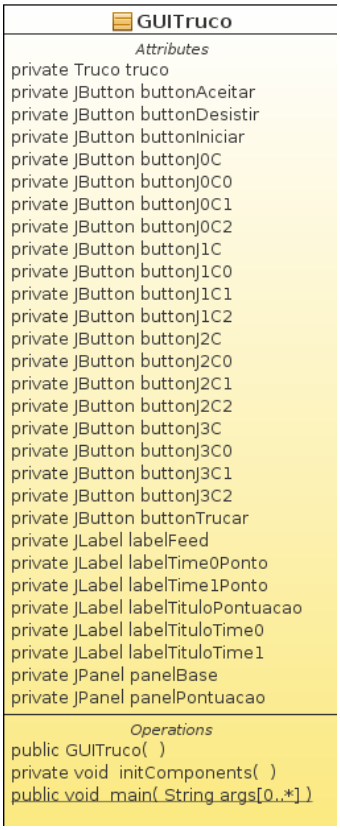

Figura 5: Interface gráfica

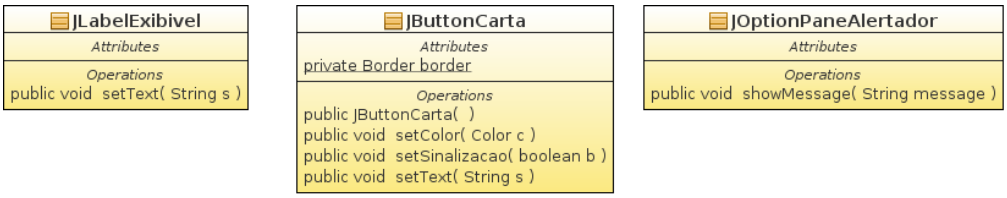

Figura 6: Implementando Interfaces de Interação

4 Conclusão

Para a implementação do jogo Trucpo aplicamos os princípios da orientação orientada a objetos e também os padrões de projeto *Singleton*, *Observer*, *State*, *Visitor*. Além disso, para a execução do código utilizamos uma interface gráfica, criada a partir do ambiente de desenvolvimento do NetBeans. A partir desse trabalho, observamos, portanto, o propósito de aplicação dos padrões de projeto *Singleton*, *Observer*, *State*, *Visitor* e *Adapter*. Além disso, os resultados finais estão de acordo com o que foi proposto. Dessa forma, pode-se dizer que o trabalho foi bem sucedido e que contribui para o aprendizado da programação modular.