

**MC322**  
Segundo semestre de 2018

**Trabalho 1**

**Professore:** Fábio Luiz Usberti (fusberti@ic.unicamp.br)  
**PED** Luis Henrique Pauleti Mendes (luishpmendes@gmail.com)  
**PAD** Pedro Barros Bastos (p204481@dac.unicamp.br)

---

## 1 Objetivo

Estudo e implementação dos conceitos de programação orientada a objetos, abordados em aula, para uma aplicação na área de jogos de cartas.

## 2 Regras do Jogo

O jogo consiste em uma variação do jogo de cartas russo **Svoyi Koziri**<sup>1</sup>, que pode ser jogado com um número de cartas múltiplo de quatro, ranqueadas em ordem crescente de 2, 3, ..., 9, 10, J, Q, K, A em cada naipe.

O jogo consiste em duas fases: uma fase de **inicialização**, na qual as cartas são distribuídas aos jogadores, e uma fase de **rodadas**, onde em cada rodada cada jogador faz sua jogada. O jogo termina quando um dos jogadores não tiver mais cartas na mão ou quando um limite de rodadas for atingido. Ganha o jogador que conseguir esvaziar sua mão. Caso o número limite de rodadas seja atingido sem nenhum jogador ter conseguido esvaziar a mão, ganha o jogador que tiver menos cartas na mão. Caso os dois jogadores tenham o mesmo número de cartas na mão, ganha o jogador que conseguir vencer o maior número de rodadas.

### 2.1 Inicialização


Chamaremos o jogador que inicia a primeira rodada de **Jogador1** e o outro de **Jogador2**.

O Jogador1 deve escolher uma cor (**vermelho** ou **preto**) e um naipe da cor escolhida para ser seu trunfo. Chamaremos a cor e o naipe trunfo do Jogador1 de **Cor1** e **Trunfo1**, respectivamente.

O Jogador2 deve escolher um naipe da cor remanescente para ser seu trunfo. Chamaremos a cor e o naipe trunfo do Jogador2 de **Cor2** e **Trunfo2**, respectivamente.








As cartas são embaralhadas e metade delas são entregues ao Jogador1, que deve reter as cartas de Cor1 e devolver as demais para o baralho. O Jogador2 deve receber as mesmas cartas do Jogador1, nos naipes correspondentes a Cor2. Por fim, o Jogador1 deve pegar as cartas remanescentes de cor Cor2 e, por sua vez, o Jogador2 deve pegar as cartas remanescentes de cor Cor1.

O exemplo abaixo ilustra a fase de inicialização com um baralho de 24 cartas (9, 10, J, Q, K, A):

- O Jogador1 escolhe a cor preta e o naipe ♠ como trunfo.
- Ao Jogador2 é atribuída a cor vermelha e ele escolhe o naipe ♦ como trunfo.
- Metade das cartas são distribuídas para Jogador1, dentre as quais apenas as de cor preta são retidas:  
 (as cartas dos naipes ♦ e ♥ são devolvidas ao baralho).

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Svoyi\\_Koziri](https://en.wikipedia.org/wiki/Svoyi_Koziri)

- O Jogador2 recebe as seguintes cartas: .
- Por fim, o Jogador1 recebe as cartas restantes de naipe  e , enquanto que o Jogador2 recebe as de naipe  e . Assim, os jogadores ficam com as seguintes cartas:
  - Jogador1: .
  - Jogador2: .








## 2.2 Rodadas

Na primeira rodada da fase de jogadas, o Jogador1 deve começar jogando uma carta qualquer. Então, o Jogador2 deve:

- jogar uma “carta melhor” vencendo a rodada, ou
- pegar todas as cartas na mesa perdendo a rodada

Uma “carta melhor” é uma carta de maior rank do mesmo naipe que a última carta na mesa, ou qualquer carta do naipe trunfo do Jogador2 (se a última carta da mesa for diferente do naipe trunfo do Jogador2). O jogador que vence a rodada atual é o primeiro a jogar na rodada seguinte.

O exemplo abaixo, que é uma continuação do exemplo da subseção anterior, ilustra algumas rodadas:

- O Jogador1 começa a primeira rodada jogando .
- O Jogador2 joga  e ganha a primeira rodada.
- O Jogador2 começa a segunda rodada jogando .
- O Jogador1 joga  e ganha a segunda rodada.
- O Jogador1 começa a terceira rodada jogando .
- O Jogador2 pega todas as cartas e perde a terceira rodada.
- O Jogador1 começa a quarta rodada jogando .
- O Jogador2 joga  e ganha a quarta rodada.

## 3 Descrição do Trabalho

É esperado do aluno que implemente uma classe **Player**, que irá analisar o jogo e tomar decisões para tentar vencer o jogo. Deste modo, uma das tarefas que se espera é que a classe **Player** implementada pelo aluno consiga ser competitiva, isto é, que vença o máximo de partidas que for possível ao jogar contra outras classes jogadoras.

A classe do jogador deve ser chamada **PlayerRAxxxxxx** (onde “xxxxxx” é o RA do aluno). A classe jogador do aluno deve herdar da classe **Player** abstrata (o conceito de classes abstratas será visto futuramente na disciplina). Dois atributos são herdados da classe **Player** abstrata:

- Suit trump: O naipe trunfo do jogador. Possui as cartas que estão na mão do jogador.

Os seguintes métodos deverão ser implementados para interagir com a Engine do jogo:

- O método construtor, que será responsável por inicializar o atributo do jogador:
  - `public PlayerRAxxxxxx (Suit trump) { ... }`
- O getter que será responsável por recuperar o atributo do jogador:
  - `public Suit getTrump() { ... }`
- O método que é responsável por escolher a jogada do jogador:
  - `public Play playRound (boolean firstToPlay, Engine engine) { ... }`

Este método recebe um valor booleano que indica se o jogador é o primeiro a jogar a rodada atual, e uma referência para o motor do jogo e deve retornar a jogada do jogador na forma de um objeto do tipo **Play**.

## 4 Descrição da Engine

O Engine irá se comunicar com a classe **PlayerRAxxxxxx** através dos métodos supracitados. O método construtor é executado somente uma vez, fornecendo o naipe trunfo do jogador. Em seguida a mão do jogador será definida e poderá ser acessada pelo jogador através do método `getHandOfPlayer()`. E, por fim, serão feitas várias chamadas ao método **playRound()**, uma para cada rodada da partida.

A seguir serão apresentadas as principais classes e enums que serão utilizadas no trabalho.

### 4.1 Enum Color

O enum Color representa uma cor de naipe e é descrito na Figura 1.

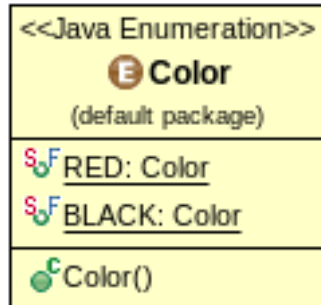


Figura 1: Enum Color.

Conforme podemos ver na Figura 1, são definidas duas cores de naipes: vermelho (RED) e preto (BLACK).

### 4.2 Enum Suit

O enum Suit representa um naipe de uma carta e é descrito na Figura 2.

Conforme podemos ver na Figura 2, são definidos quatro naipes, sendo dois vermelhos (HEARTS e TILES) e dois pretos (CLOVERS e PIKES).

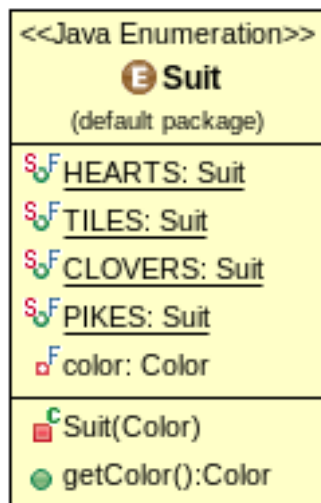


Figura 2: Enum Suit.

### 4.3 Enum Rank

O enum Rank representa um rank de uma carta e é descrito na Figura 3.

Conforme podemos ver na Figura 3, são definidos treze ranks, que correspondem aos treze ranks das cartas de baralho em ordem crescente: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A.

### 4.4 Classe Card

A classe Card representa uma carta de baralho e é descrita na Figura 4.

Conforme podemos ver na Figura 4, a classe Card possui dois atributos privados, cujos valores podem ser recuperados por meio dos métodos get:

- suit: um atributo do tipo Suit que representa o naipe da carta.
- rank: um atributo do tipo Rank que representa o rank da carta.

### 4.5 Enum PlayType

O enum PlayType representa um tipo de jogada do jogo Svoyi Koziri e é descrito na Figura 5.

Conforme podemos ver na Figura 5, são definidos dois tipos de jogada: jogar uma carta na mesa (PLAYACARD) e pegar todas as cartas da mesa (TAKEALLCARDS).

### 4.6 Classe Play

A classe Play representa uma jogada do jogo Svoyi Koziri e é descrita na Figura 6

Conforme podemos ver na Figura 6, a classe Play possui dois atributos privados, cujos valores podem ser recuperados e definidos por meio dos métodos get e set:

- type: um atributo do tipo PlayType que representa o tipo da jogada.
- card: um atributo do tipo Card que representa a carta da jogada.

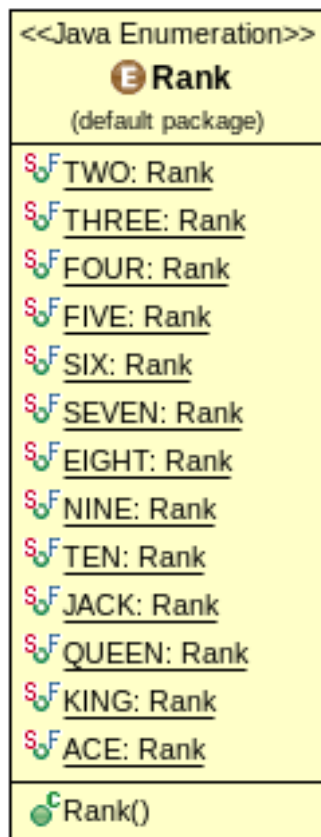


Figura 3: Enum Rank.

## 5 Exemplos de Código

Aqui mostraremos alguns exemplos de como criar um objeto **Play** dos tipos jogar uma carta e pegar todas as cartas.

Exemplo de jogada para jogar uma carta:

```

1 // card : objeto Card que quero jogar
2 Play play = new Play(PlayType.PLAYACARD, card);
  
```

Exemplo de jogada para pegar todas as cartas:

```

1 Play play = new Play(PlayType.TAKEALLCARDS);
  
```

## 6 Criação do Projeto

Os seguintes passos podem ser tomados para a criação do projeto:

1. Abra o Eclipse.
2. Crie um novo projeto ("File" -> "New -> "Project..." -> "Java Project").

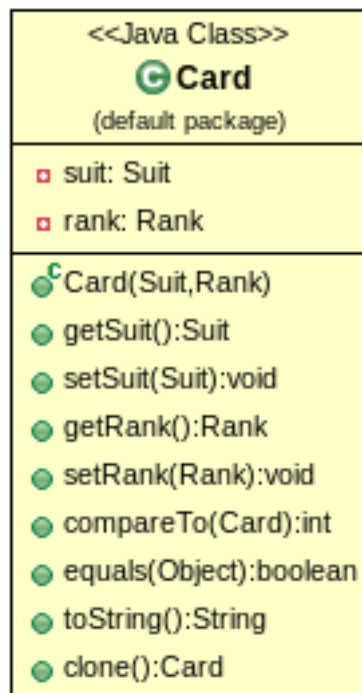


Figura 4: Classe Card.

3. Digite o nome do projeto (ex: “Trab1”).

4. Clique em “Finish”.

Para importar o jar faça:

1. Vá em “Project” -> “Properties” -> “Java Build Path”

2. Na aba “Libraries” clique em “Add External JARs...”.

3. Selecione o arquivo “svoyiKoziri.jar”.

4. Clique em “Apply and Close”.

Crie o pacote **driver** e cole a classe **Main.java** dentro do projeto no pacote driver.

Crie o pacote **player** e, dentro dele, crie a classe **PlayerRxxxxxx**, selecionando a classe Player como superclasse.

Utilize a classe Main para testar sua classe, substituindo um dos DummyPlayer pela sua classe.

## 7 Saída do programa

**Atenção:** a classe **PlayerRxxxxxx** não deve imprimir nada na saída do programa.

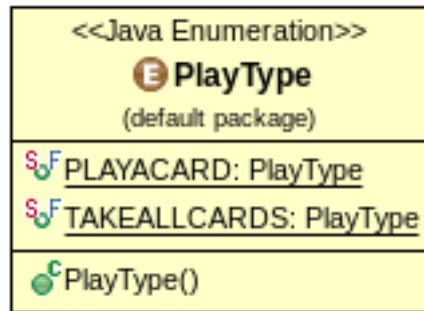


Figura 5: Enum PlayType.

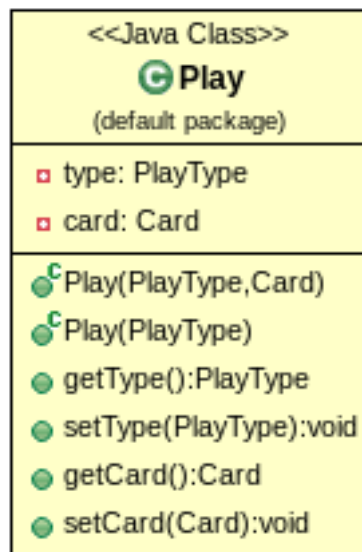


Figura 6: Classe Play.

## 8 Critério de Avaliação

A nota do trabalho será composta por dois componentes, conforme mostra a equação a seguir:

$$T_1 = \min\{Q_1 + C_1, 10\}$$

Onde:

- $T_1$ : Nota do Trabalho 1
- $Q_1 \in [0, 10]$ : Componente de qualidade do trabalho de acordo com os critérios: corretude, aderência, estratégia, comentários e convenções.
- $C_1 \in [0, 2]$ : Componente de competitividade do jogador, ou seja, o quão eficiente é esse jogador ao disputar com os demais jogadores. Cabe observar que este componente entrará como **bonificação** à nota do Trabalho 1.

## 8.1 Componente de qualidade da implementação

1. **Corretude:** O código não deve possuir *warnings* ou erros de compilação. O código não deve emitir *exceptions* em nenhuma situação. As jogadas realizadas pelo jogador devem ser válidas, segundo as regras do jogo, em todas as situações.
2. **Aderência ao Enunciado:** A implementação deve realizar o que é requisitado no enunciado.
3. **Documentação:** Os comentários devem ser suficientes para explicar os trechos mais importantes da implementação, utilizando-se de terminologias corretas vistas em sala de aula quando isso se aplicar. Faça, também, a documentação em Javadoc da sua classe, explicando detalhadamente cada atributo e método.
4. **Estratégia do jogador:** No comentário Javadoc no cabeçalho do código-fonte deve haver um texto, de pelo menos 15 e no máximo 50 linhas, descrevendo organizadamente a estratégia adotada pelo seu jogador, ou seja, quais critérios são utilizados para escolher a jogada a ser realizada em cada rodada. (Obs: a organização e clareza do seu texto também implicará na nota deste componente).
5. **Convenções:** Os nomes das entidades de seu código devem seguir as convenções Java. Além disso, seu código deve estar corretamente indentado e bem apresentado segundo as boas práticas da linguagem.

## 8.2 Componente de competitividade do jogador

O componente de competitividade refere-se ao número de vitórias do jogador relativo ao número de vitórias dos demais jogadores. O jogador mais competitivo (maior número de vitórias) receberá a bonificação máxima (2). Será introduzido um jogador “ingênuo” no campeonato que fornecerá um limitante inferior de competitividade. Os jogadores menos competitivos (que tiverem um número de vitórias menor ou igual ao do jogador “ingênuo”) receberão a bonificação mínima (0). O componente de competitividade será calculado de acordo com a seguinte equação:

$$C_1 = \min \left\{ 2 \times \left( \frac{V_{jogador} - V_{min}}{V_{max} - V_{min}} \right), 0 \right\}$$

- $V_{jogador}$ : Número de vitórias obtidas pelo jogador no campeonato.
- $V_{min}$ : Número de vitórias obtidas pelo jogador “ingênuo” do campeonato.
- $V_{max}$ : Número máximo de vitórias obtidas pelo melhor jogador do campeonato.

## 9 Observações

- O trabalho é individual.
- Não é permitido nenhum tipo de compartilhamento de código entre os alunos ou o uso de códigos de terceiros. Uma única exceção permitida consiste nas APIs da linguagem Java. Em caso de plágio, todos os alunos envolvidos serão reprovados com média zero.
- Não é permitido explorar nenhuma brecha de segurança (*exploit*) da Engine. Se você encontrar alguma brecha, relate imediatamente ao professor.



## **10 Submissão**

O trabalho deverá ser submetido até as 23:59 do dia 01 de outubro de 2018, através do Moodle. A submissão deve ser exclusivamente um arquivo nomeado JogadorRAxxxxxx.java (onde “xxxxxx” é o RA do aluno). Após submeter, certifique-se de que o arquivo enviado é o correto.