Programação Estruturada

Aula 14 - Adicionando novos recursos ao jogo da velha

Apresentação

Na aula anterior, no intuito de aplicar os conhecimentos que adquirimos ao longo de nosso curso, desenvolvemos um Jogo da Velha. Você aprendeu a implementar o jogo básico, com instanciação de jogadores e das posições de marcação no tabuleiro.

Nesta aula, deixaremos nosso jogo ainda mais interessante. Incluiremos novas funcionalidades. Uma dessas funcionalidades é a contagem do tempo da partida.

Ao final do jogo, será informado aos jogadores quanto tempo durou a partida. Além disso, implementaremos o cadastro dos jogadores e o ranking.

Os nomes dos jogadores ficarão armazenados para, ao final do jogo, ser realizada a contagem de partidas de que participaram e a quantidade de vitórias de cada um. Esse ranking será armazenado em um arquivo e exibido ao final de cada partida.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Saber incluir funcionalidades de cadastro de jogadores, contagem de tempo da partida e geração de arquivo de ranking ao jogo da velha desenvolvido na aula anterior.
- Aplicar as novas funcionalidades ao jogo de damas.

1. Cadastro de Jogadores

Agora, vamos adicionar ao nosso jogo uma funcionalidade que permitirá o cadastro dos nomes dos jogadores. Para isso, iremos escrever uma rotina que nomearemos “cadastro”, em que serão solicitados e, posteriormente, armazenados os nomes dos jogadores.

Antes de escrever a rotina, porém, precisamos criar mais duas variáveis na classe JogoDaVelha para armazenar os nomes dos jogadores. No início do código, no local onde você declarou as outras variáveis do jogo básico, declare as seguintes variáveis:

```
1 private static String jogador1, jogador2;
```

Agora, podemos criar o nosso procedimento de cadastro. Veja o código da rotina, a seguir:

```
1 public static void cadastro() {
2    System.out.println("Digite o nome do jogador 1:");
3    //recebe nome do jogador 1
4    jogador1 = leitor.next();
5    System.out.println("Digite o nome do jogador 2:");
6    //recebe nome do jogador 2
7    jogador2 = leitor.next();
8 }
```

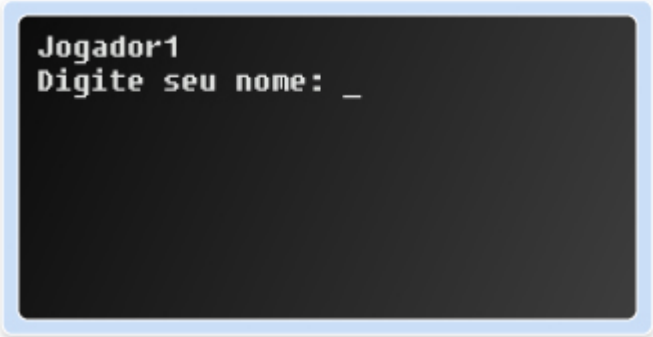
Veja que o procedimento “cadastro”, por enquanto, é bem simples. O programa apenas solicita a entrada dos nomes dos jogadores e, em seguida, captura os nomes digitados para armazenar nas variáveis globais que criamos lá na parte superior do código, variáveis estáticas “jogador1” e “jogador2”.

Antes de incrementarmos um pouco mais o jogo, vamos fazer algumas alterações no jogo que fizemos na aula anterior? Para que o cadastro dos jogadores funcione, precisamos primeiro chamar o procedimento de cadastro, dentro da rotina principal. Vá até a rotina main e adicione no início da rotina a seguinte linha de código que chama o procedimento cadastro:

```
1 cadastro();
```

Agora, o cadastro já está funcionando. Você já pode testar. Compile e execute o jogo. Veja que, antes de mostrar o tabuleiro do jogo, aparece a tela que vemos na Figura 1.

Figura 01 - Solicitação de nome de jogador na tela inicial do jogo



Ao digitar o nome do jogador 1, será solicitado o nome do segundo jogador e o jogo será iniciado.

Observe, porém, que, ao indicar de quem é a vez, a mensagem que aparece ainda faz referência aos jogadores como “jogador 1” e “jogador 2” e não aos seus respectivos nomes (que acabaram de ser cadastrados). Isso ocorre porque, na aula anterior, o nosso código ainda não “conhecia” os jogadores pelos nomes (não havia cadastro). Vamos mudar isso?

No procedimento “jogar(int jogador)”, veja que, dentro de seu corpo, temos uma linha de código com um “System.out.println” que mostra de quem é a vez de jogar. Essa linha faz referência a um inteiro (jog) que está inicializado em uma estrutura “if” lá no início da função:

```
1 // definindo o jogador da vez
2 if (jogador == 1) {
3    jog = 1;
4 } else {
5    jog = 2;
6 }
7 System.out.println("\n\n Vez do Jogador " + jog);
```

A partir de agora, o nosso jogo não vai mais se referir aos jogadores como “jogador 1” e “jogador 2”, e sim como seus nomes cadastrados. Dessa forma, precisamos alterar o código, trocando o “System.out.println” e a estrutura do “if-else” mostrados pelo seguinte código:

```
// definindo o jogador da vez
if (jogador == 1) {
    jog = 1;
    System.out.println("\n\n Vez do Jogador " + jogador1);
} else {
    jog = 2;
    System.out.println("\n\n Vez do Jogador " + jogador2);
}
```

O código acima mostra de quem é a vez de jogar, referindo-se ao jogador pelo nome que ele cadastrou. Vamos ver como fica o código completo da função jogar, feitas essas alterações:

```
public static void jogar(int jogador) {
   // inicializando contador da estrutura while
   int i = 0;
   // definindo o jogador da vez
   if (jogador == 1) {
      jog = 1;
      System.out.println("\n\n Vez do Jogador " + jogador1);
   } else {
      jog = 2;
      System.out.println("\n\n Vez do Jogador " + jogador2);
   }
   while (i == 0) {
      linha = 0; // inicializando valor da linha
      coluna = 0; // inicializando valor da coluna
      while (linha < 1 || linha > 3) {
         System.out.print("Escolha a Linha (1,2,3):");
         // lendo a linha escolhida
         linha = leitor.nextInt();
         // Aviso de linha inválida, caso o jogador tenha
         // escolhido linha menor que 1 ou maior que 3
         if (linha < 1 || linha > 3) {
            System.out.println("Linha invalida! Escolha uma linha entre 1 e 3.");
         }
      }
      while (coluna < 1 || coluna > 3) {
         System.out.print("Escolha a Coluna (1,2,3):");
         // lendo a coluna escolhida
         coluna = leitor.nextInt();
         if (coluna < 1 || coluna > 3) {
            System.out.println("Coluna invalida! Escolha uma linha entre 1 e 3.");
         }
      }
      // Ajusta índices para começar do zero
      linha = linha - 1;
      coluna = coluna - 1;
      if (casa[linha][coluna] == 0) {
         // se não estiver marcado
         // marcar com o símbolo do jogador da vez
         casa[linha][coluna] = jog;
      i = 1;
      } else { // se o campo escolhido já estivar marcado
         System.out.println("Posição ocupada!");
      }
   }
}
```

Agora, teste mais uma vez e veja o resultado. Jogue até o fim e veja que, ao anunciar o vencedor, o jogo ainda não está mostrando os nomes dos jogadores. Vamos alterar isso também?

Na rotina “main”, veja o seguinte código:

```
if (win == 1 || win == 2) {
    // informa o vencedor
    System.out.println("Jogador " + win + " é o ganhador!");
} else {
    // se não houve vencedor
    System.out.println("Não houve vencedor! O jogo foi empate!!");
}
```

A variável “win” foi declarada como *int*, para mostrar o vencedor como jogador 1 ou jogador 2. Para mostrarmos o nome do vencedor, precisamos alterar isso. Mude o código para:

```
if (win == 1) {
    // informa o vencedor
    System.out.println("Jogador " + jogador1 + " é o ganhador!");
} else if (win == 2) {
    // informa o vencedor
    System.out.println("Jogador " + jogador2 + " é o ganhador!");
} else {
    // se não houve vencedor
    System.out.println("Não houve vencedor! O jogo foi empate!!");
}
```

Veja que precisamos inserir duas estruturas “if” para verificar cada um dos jogadores pelo nome, não mais por um número.

Agora, você já pode testar. Compile e execute. Veja que, dessa vez, após cadastrar os jogadores no início do jogo, cada um é chamado pelo nome durante a partida, seja ao mostrar de quem é a vez, ou ao dizer quem é o vencedor.

Vídeo 02 - Cadastro

Atividade 01

1. Implemente e teste a função cadastro para o jogo da velha, conforme mostrado nesta aula.
2. Implemente a função cadastro para o jogo de damas que você iniciou nas atividades da aula anterior.

2. Contagem de Tempo da Partida

Agora que já temos a função de cadastro implementada, outra funcionalidade interessante a adicionar ao jogo da velha seria a contagem do tempo da partida. Com a adição dessa funcionalidade, o tempo seria exibido ao final da partida. A implementação é simples. Vamos ver?

Primeiro, precisamos criar uma variável do tipo *long*. Essa variável vai guardar o tempo inicial da partida.

Insira a seguinte linha de código, logo abaixo de onde você declarou as variáveis globais(no início do código):

```
1 private static long inicio;
```

Depois, vamos criar uma rotina do tipo *void* para armazenar o tempo inicial da partida na variável “inicio”. Vamos então criar uma rotina chamada “setTempo”. Veja:

```
1 public static void setTempo() {
2    inicio = System.currentTimeMillis();
3 }
```

Essa rotina usa a função System.currentTimeMillis() para retornar um *long* que representa a data/hora atual em termos de milissegundos. Basicamente, a ideia é representar a data/hora atual em termos de quantos milissegundos se passaram desde a meia noite de primeiro de janeiro de 1970.

Em seguida, precisamos criar uma função que retorne aos jogadores o valor do tempo atual, no fim da partida. Esse valor se dá pela diferença entre o tempo inicial (armazenado na variável “inicio”) e o tempo atual, representado por System.currentTimeMillis(). Veja como fica a função “getTempo”, que faz e nos retorna esse cálculo.

```
1 public static long getTempo() {
2     return (System.currentTimeMillis() – inicio)/1000;
3 }
```

Simples, não é mesmo? Está quase pronto. Falta apenas chamarmos as duas rotinas dentro da rotina principal “main”. Primeiro, você chama a função setTempo() para iniciar a contagem. Faça isso no início da rotina “main”, logo depois da chamada à função “cadastro”. Assim:

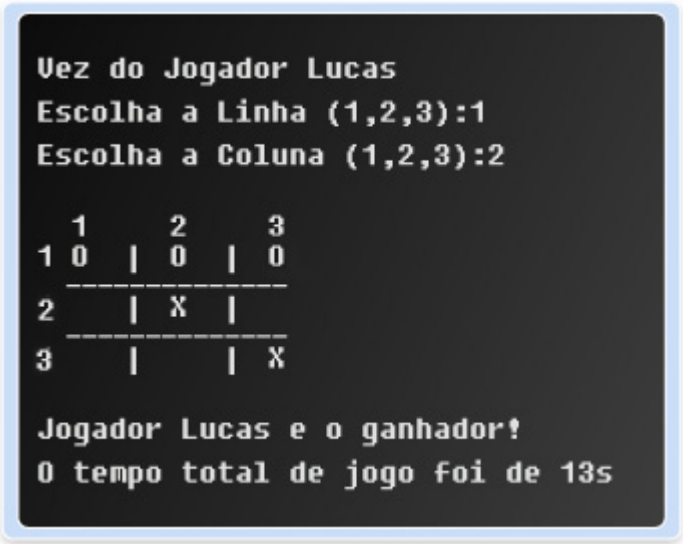
```
1 cadastro(); // já existe no código
2 setTempo(); // linha que você vai inserir
3 ...
```

Já no final da rotina main, devemos colocar o seguinte código:

```
1 System.out.println("O tempo total de jogo foi de " + getTempo() + "s ");
```

Na linha de código que acabamos de inserir, estamos informando o tempo da partida, através da chamada à função getTempo(). Pronto! Está finalizada a implementação da contagem de tempo. Veja, na Figura 2, o tempo de partida sendo exibido ao final da partida.

Figura 02 - Exibição do tempo de partida



Agora, você já pode testar! Compile e execute o código e veja em quanto tempo termina a partida.

Vídeo 03 - Cálculo do Tempo

Atividade 02

1. Implemente a funcionalidade de contagem de tempo no jogo da velha, conforme mostrado em aula.
2. Implemente a funcionalidade de contagem de tempo no jogo de damas.

3. Histórico de Vitórias e Derrotas – Implementação do Ranking

Para incrementar ainda mais o nosso jogo, vamos inserir um ranking, onde poderemos visualizar a quantidade de vitórias e derrotas de cada jogador que foi cadastrado.

Para implementar essa funcionalidade, precisamos encontrar uma maneira de armazenar a quantidade de vitórias e derrotas de cada jogador. Para isso, temos que declarar duas variáveis do tipo “int” (uma para armazenar a quantidade de vitórias e outra para armazenar a de derrotas) e associá-las ao nome do jogador. Como fazemos isso? Utilizando um registro.

Veja o seguinte código, que declara uma classe para representar o registro a ser criado:

```
1 public class Jogador {
2    public String nome;
3    public int vitorias;
4    public int derrotas;
5 }
```

Muito bem, agora precisamos indicar ao nosso código a forma com que essas informações serão acessadas para adicionar, a cada partida, os nomes e quantidades de vitórias e derrotas ao arquivo que criamos. Para isso, primeiro temos que fazer algumas alterações no código. A primeira delas será a de informar que utilizaremos o tipo de dados registro chamado de Jogador.

Faremos isso substituindo os dados que tínhamos anteriormente (só o nome) pelo registro. Dessa forma, trocamos a definição existente das seguintes variáveis:

```
1 private static String jogador1, jogador2;
```

pelas seguintes declarações:

```
1 private static Jogador jogador1, jogador2;
```

Essa alteração indica que queremos não só usar o nome dos jogadores (tipo String), mas todos os demais dados (quantidade de vitórias e de derrotas). Se você fizer essa alteração em seu código, você irá notar que ela irá gerar vários erros de compilação no código, mais especificamente na função cadastro(). Isso porque mudamos o tipo de dados das variáveis jogador1 e jogador2. Para corrigir esses problemas, basta substituir o código anterior da função cadastro.

```
public static void cadastro() {
    System.out.println("Digite o nome do jogador 1:");
    //recebe nome do jogador 1
    jogador1 = leitor.next();
    System.out.println("Digite o nome do jogador 2:");
    //recebe nome do jogador 2
    jogador2 = leitor.next();
}
```

pelo seguinte código:

```
public static void cadastro() {
    System.out.println("Digite o nome do jogador 1:");
    //recebe nome do jogador 1
    String nome_jogador1 = leitor.next();
    jogador1 = buscarJogador(nome_jogador1);
    System.out.println("Digite o nome do jogador 2:");
    //recebe nome do jogador 2
    String nome_jogador2 = leitor.next();
    jogador2 = buscarJogador(nome_jogador2);
}
```

Como você pode notar, agora criamos duas variáveis locais (**nome_jogador1** e **nome_jogador2**) do tipo String para receber o nome dos jogadores 1 e 2. Em seguida, a cada leitura de nome, inicializamos os registros jogador1 e jogador2, de acordo com os nomes lidos. Nesse caso, precisamos verificar se esses jogadores já jogaram o jogo da velha e, se for esse o caso, precisamos recuperar os dados históricos das partidas deles (quantidades de vitórias e de derrotas). Esse é o objetivo da função **buscarJogador()**, o qual será mostrado mais adiante.

Além dessas modificações, precisamos trocar também o uso de jogador1 e jogador2 por jogador1.nome e jogador2.nome, respectivamente, como no caso de

```
System.out.println("Jogador " + jogador1 + " é o ganhador!");
```

que trocamos por:

```
System.out.println("Jogador " + jogador1.nome + " é o ganhador!");
```

Essas trocas acontecem nas rotinas main() e jogar().

Mas como implementar a função buscarJogador(), se os dados de vitória ou derrota não estão sendo salvos em nenhum lugar? Vamos resolver esse detalhe antes de mostrar o código de buscarJogador(), ok? Para armazenar os registros, vamos usar um array, como mostrado a seguir:

```
1  public class JogoDaVelha {
2      private static Jogador[] jogadores = new Jogador[50];
3      private static int quantidadeJogadores = 0;
4      ...
5  }
```

O array **jogadores** será utilizado para guardar todos os jogadores, até o limite de 50. Para saber quantos jogadores estão cadastrados no array, utilizamos a variável **quantidadeJogadores**. Vamos agora ver o que precisamos fazer para buscar e guardar os dados sobre jogadores no array, através da função buscarJogador():

```
1  public static Jogador buscarJogador(String nome) {
2      Jogador jogador = null;
3      for (int i = 0; (i < quantidadeJogadores) && (jogador == null); i++) {
4          if (jogadores[i] != null &&
5              jogadores[i].nome.equalsIgnoreCase(nome)) {
6              jogador = jogadores[i];
7          }
8      }
9      if (jogador == null) {
10         jogador = new Jogador();
11         jogador.nome = nome;
12         if (quantidadeJogadores < 50) {
13             jogadores[quantidadeJogadores] = jogador;
14             quantidadeJogadores = quantidadeJogadores + 1;
15         }
16
17     }
18     return jogador;
19 }
```

Note na função a declaração da variável **jogador**, responsável por armazenar o registro a ser retornado. Além dela, temos um laço for, o qual percorre os registros armazenados no array da posição 0 (zero) até a quantidade de jogadores (último índice utilizado no array + 1). Lembrando que, no início, o laço não será executado, visto que a quantidade de jogadores inicial é igual a zero. Outra condição para o laço ser executado é que o valor da variável "jogador" seja igual a **null**, valor que representa um registro não criado (não encontrado), ou seja, um valor vazio.

Dentro do laço for, usamos a expressão “jogadores[i]” para acessar o registro da posição “i” do array “jogadores”. A partir desse registro, fazemos duas verificações: se a posição acessada no array não está vazia (valor diferente de null); se o nome do jogador do registro acessado (**jogadores[i].nome**) é igual ao nome passado como parâmetro para a função. Se ambas as verificações forem confirmadas, isto quer dizer que encontramos o registro do jogador! Com isso, é só atribuí-lo à variável “jogador”.

Após a execução do laço, verificamos se a variável “jogador” é nula. Se isso acontecer, é porque não existe ainda um registro cadastrado no array para o nome passado como parâmetro. Dessa forma, dentro desse if temos a criação de um novo registro e a inicialização de seu nome (“jogador.nome”) com o nome passado como parâmetro para a função. Feito isso, adicionamos o registro criado na última posição do array (“quantidadeJogadores”), caso haja espaço disponível no array (**quantidadeJogadores < 50**), aumentando assim em uma unidade o valor da variável “quantidadeJogadores”.

Terminado esse processo, podemos então retornar o valor da variável “jogador”, o qual representa agora o registro utilizado para representar o jogador com o nome informado (parâmetro da função).

Essa função, utilizada dentro da função cadastro() é responsável por criar novos registros e por recuperar registros já existentes. Falta agora mostrar para o usuário a lista de jogadores e respectivos resultados de partidas (vitórias e derrotas). Isso é feito através da rotina imprimirJogadores():

```
1 public static void imprimirJogadores() {
2     System.out.println("------ Resultado dos jogadores --------");
3     for (int i = 0; i < quantidadeJogadores ; i++) {
4         System.out.println("Nome: " + jogadores[i].nome);
5         System.out.println(" | vitórias:" + jogadores[i].vitorias);
6         System.out.println(" | derrotas:" + jogadores[i].derrotas);
7     }
8 }
```

Note que a função imprimirJogadores percorre o array, imprimindo os resultados de vitórias e derrotas de cada jogador cadastrado. Essa rotina deve ser chamada na última linha da rotina main(), a qual deve possuir os ajustes para registrar as vitórias e derrotas de cada jogador:

```
public static void main(String[] args) {
   cadastro();
   setTempo();
   int i = 0;
   // percorre todo o tabuleiro, nas nove posições:
   for (i = 0; i < 9; i++) {
      jogo();// chama a função jogo(), que desenha o tabuleiro
      if (i % 2 == 0) {
         jogar(2);
      } else {
         jogar(1);
      }
      // chama a função check (), para ver se alguém ganhou
      check();
      if (win == 1 || win == 2) {
         // sai do laço antes de completar o tabuleiro,
         // se alguém tiver vencido
         i = 10;
      }
   }
   // chama a função jogo(), para desenhar novamente o tabuleiro
   jogo();
   // verifica se houve vencedor
   System.out.println();
   if (win == 1) {
      // informa o vencedor
      System.out.println("Jogador " + jogador1.nome + " é o ganhador!");
      jogador1.vitorias = jogador1.vitorias + 1;
      jogador2.derrotas = jogador2.derrotas + 1;
   } else if (win == 2) {
      // informa o vencedor
      System.out.println("Jogador " + jogador2.nome + " é o ganhador!");
      jogador2.vitorias = jogador2.vitorias + 1;
      jogador1.derrotas = jogador1.derrotas + 1;
   } else {
      // se não houve vencedor
      System.out.println("Não houve vencedor! O jogo foi empate!!");
   }
   System.out.println("O tempo total de jogo foi de " + getTempo() + "s");
   imprimirJogadores();
}
```

Vídeo 04 - Uso do Registro

Atividade 03

1. Implemente as funcionalidades de ranking para o jogo da velha, conforme mostrado em aula.
2. Implemente a funcionalidade do ranking para o jogo de damas.

5. Salvando Dados em Arquivo

Agora que já criamos um tipo de dado do tipo registro para representar os dados de um jogador e que implementamos as funcionalidades básicas para um rank, precisamos definir em qual arquivo vão ser guardadas as informações de todos os jogadores, para que as mesmas não sejam perdidas entre uma partida e outra. Veja a seguinte declaração, que precisa ficar no início do corpo da classe JogoDaVelha:

```
1 private static File arquivo = new File("ranking.obj");
```

A variável **“arquivo”** é utilizada para representar o arquivo a ser criado para armazenar os dados do rank. Lembre-se que, para usar as classes Java que manipulam arquivos, precisamos importá-las através de comandos como o seguinte:

```
1 import java.io.File;
```

Para cada classe utilizada, você deve ter certeza de que o seu respectivo comando de **import** foi utilizado. Para implementar a manipulação de arquivos (escrita e leitura), vamos criar duas novas rotinas. A primeira delas é a **salvarJogadores()**, como mostrado a seguir:

```
1 public static void salvarJogadores() {
2     try {
3         ObjectOutputStream saida = new ObjectOutputStream(new FileOutputStream(arquivo));
4         saida.writeObject(jogadores);
5     } catch (Exception e) {
6         throw new RuntimeException(e);
7     }
8 }
```

O código mostrado faz uso das classes **ObjectOutputStream** e **FileOutputStream** de Java. Ao criar um ObjectOutputStream, você deve passar um FileOutputStream que referencia o arquivo a ser criado (variável “arquivo”). Após isso, para salvar os registros basta chamar a rotina **saida.writeObject()** passando como parâmetro o array de registros “jogadores”. Não é fácil? Agora, todo esse código precisa estar dentro de um bloco *try/catch*, o qual você estudará bem no módulo de orientação a objetos, ficando nesta aula apenas o seu uso sem maiores explicações, já que isso vai além do contexto de nosso módulo.

Além da rotina para salvar registros, precisamos também de uma rotina para lê-los, chamada de **lerJogadores()**:

```
1  public static void lerJogadores() {
2      try {
3          ObjectInputStream saida = new ObjectInputStream(new FileInputStream(arquivo));
4          jogadores = (Jogador[]) saida.readObject();
5          while (jogadores[quantidadeJogadores] != null && quantidadeJogadores < 50) {
6              quantidadeJogadores = quantidadeJogadores + 1;
7          }
8      } catch (FileNotFoundException e) {
9          // Não faz nada
10     } catch (Exception e) {
11         throw new RuntimeException(e);
12     }
13 }
```

Na rotina lerJogadores(), é preciso fazer uso das classes ObjectInputStream e FileInputStream, que funcionam de forma similar ao que foi mostrado na rotina salvarJogadores(), só que agora a função é de leitura, e não escrita. A leitura é feita através da função **saida.readObject()**. Como essa função é genérica (funciona para vários tipos de dados), precisamos especificar o tipo que sabemos estar contido no arquivo, o qual é **Jogador[]**. Especificado esse tipo, podemos atribuir o resultado à variável “jogadores”. Por fim, precisamos atualizar a variável quantidadeJogadores de acordo com a quantidade de jogadores retornado dentro do vetor!

Para o jogo funcionar em arquivo, precisamos fazer só mais alguns ajustes: chamar a rotina lerJogadores() no início da main; chamar a rotina salvarJogadores no final da main; alterar a classe Jogador da seguinte forma:

```
1  public class Jogador implements Serializable {
```

Essa alteração na classe Jogador é para que as rotinas que utilizamos (writeObject e readObject) possam funcionar, sendo que esse assunto vai ser aprofundado no módulo de orientação a objetos. E se você adicionou corretamente os *import* às classes que utilizamos, a lista de *import* do arquivo JogoDaVelha estará da seguinte forma:

```
1 import java.io.File;
2 import java.io.FileInputStream;
3 import java.io.FileNotFoundException;
4 import java.io.FileOutputStream;
5 import java.io.ObjectInputStream;
6 import java.io.ObjectOutputStream;
7 import java.util.Scanner;{
```

Já o arquivo Jogador possuirá o seguinte *import*:

```
1 import java.io.Serializable;
```

Agora sim! O nosso jogo está completo com as funcionalidades básicas que implementamos na aula anterior e os recursos de cadastro, contagem de tempo e ranking. Compile, execute o jogo e divirta-se!

Atividade 04

---

1. Implemente as funcionalidades de salvar rank em arquivo para o jogo da velha, conforme mostrado em aula.
2. Implemente a funcionalidade de salvar rank em arquivo para o jogo de damas.

5. Resumo

Nesta aula, você aprendeu a inserir novos recursos ao jogo da velha. Primeiro, você aprendeu a criar a funcionalidade de cadastro de jogadores, em que os nomes de cada jogador são armazenados no jogo. Depois, você implementou a contagem do tempo de partida, para que o tempo fosse exibido ao final da partida. Finalmente, você construiu o registro de vitórias e derrotas dos jogadores, para ser exibido, também, ao final da partida.

6. Autoavaliação

1. Descreva os principais recursos que você utilizou para implementar as seguintes funcionalidades:
    a. Cadastro de jogadores.
    b. Contagem de tempo da partida.
    c. Registro de vitórias e derrotas dos jogadores.
    d. Salvar e ler os registros em arquivo.

7. Referências

JOGO da velha: entendendo um pouco mais da linguagem C: tutorial na internet. Disponível em: <http://ube-164.pop.com.br/repositorio/12329/meusite/jogo_da_velha.pdf>. Acesso em: 20 jan. 2010.

RADTKE, Paulo V. W. **Jogo da velha**: projeto. Curitiba: PUC Paraná, 2006. Disponível em: <http://www.ppgia.pucpr.br/~radtke/jogos/velha/projeto-jogo_da_velha.pdf>. Acesso em: 20 jan. 2010.

Ana Fernanda Gomes Ascencio, Edilene Aparecida Veneruchi de Campos. FUNDAMENTOS DA PROGRAMAÇÃO DE COMPUTADORES: ALGORITMOS, PASCAL, C/C++ E JAVA. Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.