Extra

```
import java.util.Scanner;

public class Matrix {
   public static void main(String[] args) {
      boolean continuar = true;
      char opcao;
      Scanner entrada = new Scanner(System.in);

      while(continuar){
         System.out.println("Você está na matrix;");
         System.out.print("Digite o caractere especial para sair da matrix: ");
         opcao = entrada.next().charAt(0);

         if(opcao=='j'){
            continuar=false;
            System.out.println("Parabéns! Você conseguiu sair da Matrix!");
         }
         else{
            System.out.println("Você não está autorizado a sair da Matrix. Estude Java.");
         };
      }
   }
}
```

Uma pequena brincadeira para descontrair

Exceções

- Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.
- Vamos ver como JVM age ao se deparar com situações inesperadas , como divisão por zero ou acesso a um índice de array(matriz) que não existe

```
public class TestaErro {
public static void main(String[] args) {
System.out.println("inicio do main");
metodo1();
System.out.println("fim do método main"); }
private static void metodo1() {
System.out.println("inicio do método 1");
metodo2();
System.out.println("fim do método 1"); }
private static void metodo2() {
System.out.println("inicio do método 2");
int[]numero=new int[10]; // matriz de dez elementos
for (int i = 0; i <=15; i++) {
numero[i]=i;
System.out.println(i); }
System.out.println("fim do método 2");
} }
```

Exceções

O programa executou bem até mostrar o índice 9 (10 elementos):

- inicio do main
- inicio do método 1
- inicio do método 2
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

Provavelmente deu este erro:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
at Erros.TestaErro.metodo2(TestaErro.java:20)
at Erros.TestaErro.metodo1(TestaErro.java:12)
at Erros.TestaErro.main(TestaErro.java:6) /* linha de
                                          chamada do método*/
```

Esta saída é conhecida como rastro de pilha(*stacktrace*).

Exceções

- Quando uma exceção é lançada(throw), a JVM entra em estado de alerta e vai ver se o método atual toma alguma precaução ao tentar executar esse trecho de código. Neste caso o método 2 nã toma nenhuma medida diferente do que vimos agora .
- Como o método2 não esta tratando o problema, JVM para a execução dele anormalmente, sem esperar ele terminar, e volta um stackframe (método1) para baixo, onde será feita nova verificação. Caso o método1 não estiver tratando problema , então ele volta para o método main, q qual também não esta tratando o problema .Aí a JVM morre(o programa termina).

Utilizando try/catch

- Modifique o Método 2:
- **private static void metodo2()**
- **{**
- System.***out.println("inicio do método 2");***
- **int[]numero=new int[10];**
- **try {**
- **for (int i = 0; i <=15; i++) {**
- numero[i]=i;
- System.***out.println(i);*** **}**
- **}catch(ArrayIndexOutOfBoundsException e){**
- System.***out.println("erro: "+ e);***
- **}**
- **}**

Exceções

Veja foi mostrado o erro e o Java finalizou o programa voltando para o método1 e main

- inicio do main
- inicio do método 1
- inicio do método 2
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- erro: java.lang.ArrayIndexOutOfBoundsException: 10
- fim do método 2
- fim do método 1
- fim do método main

- Modifique o Método 2:
- **private static void metodo2()**
- **{**
- System.***out.println("inicio do método 2");***
- **int[]numero=new int[10];**
- **for (int i = 0; i <=15; i++) {**
- **try {**
- numero[i]=i;
- System.***out.println(i);***
- **}catch(ArrayIndexOutOfBoundsException e){**
- System.***out.println("erro: "+ e);***
- **} } }**

Exceções

- inicio do main
- inicio do método 1
- inicio do método 2
- 0
- .
- .
- .
- 9
- erro: java.lang.ArrayIndexOutOfBoundsException: 10
- erro: java.lang.ArrayIndexOutOfBoundsException: 11
- erro: java.lang.ArrayIndexOutOfBoundsException: 12
- erro: java.lang.ArrayIndexOutOfBoundsException: 13
- erro: java.lang.ArrayIndexOutOfBoundsException: 14
- erro: java.lang.ArrayIndexOutOfBoundsException: 15
- fim do método 2
- fim do método 1
- fim do método main

Como o tratamento do erro esta dentro do for
Toda vez que passa pelo try mostra o erro e
o índice do array

Exceções

- Vamos colocar try/catch agora em volta da chamada do *metodo2*
- **private static void metodo1() {**
- System.***out.println("inicio do método 1");***
- **try{**
- *metodo2();*
- **}catch(ArrayIndexOutOfBoundsException e){**
- System.***out.println("erro: "+ e);***
- **}**
- System.***out.println("fim do método 1");***
- **}**

- A saída após a impressão do índice 9 :
- erro: java.lang.ArrayIndexOutOfBounds Exception: 10
- fim do método 1
- fim do método main
- Como o printf “fim do método 1” estava fora do try/catch ele rodou.

Exceções

- Vamos colocar try/catch agora em volta da chamada do *metodo2*
- **private static void metodo1() {**
- System.***out.println("inicio do método 1");***
- **try{**
- *metodo2();*
- **}catch(ArrayIndexOutOfBoundsException e){**
- System.***out.println("erro: "+ e);***
- **}**
- System.***out.println("fim do método 1");***
- **}**

- A saída após a impressão do índice 9 :
- erro: java.lang.ArrayIndexOutOfBounds Exception: 10
- fim do método 1
- fim do método main
- Como o printf “fim do método 1” estava fora do try/catch ele rodou.

Manipulação de arquivos

- Criando um arquivo de texto
Para criar uma saída formatada, vamos usar um objeto da classe Formatter. Além de abrir o arquivo para escrita, vamos testar antes para ver se é possível escrever nele, se temos permissões e se o arquivo existe.

Isso é importante para deixar nossas aplicações bem robustas e a prova de erros. Não vá, simplesmente, abrindo os arquivos. Antes, teste se eles existem, se o usuário tem permissão de abrir, para checar se é possível mesmo escrever nele.

CriandoArquivoTexto.java

```
public class CriandoArquivoTexto
{
    public static void main(String[] args)
    {
        EscreverMetodos teste = new EscreverMetodos();
        teste.abrir();
        teste.escrever();
        teste.fechar();

    }

}
```

Manipulação de arquivos

EscreverMetodos.java

```
import java.util.Formatter;
import java.util.NoSuchElementException;
import java.util.FormatterClosedException;
import java.lang.SecurityException;
import java.io.FileNotFoundException;


public class EscreverMetodos
{
    private Formatter arquivo;

        public void abrir()
        {
            try
            {
                arquivo = new Formatter("ProgramacaoProgressiva.txt");
            }
            catch( SecurityException semPermissao)
            {
                System.err.println(" Sem permissao para escrever no arquivo ");
                System.exit(1); //exit(0) é sucesso, outro número significa que terminou com problemas
            }
            catch( FileNotFoundException arquivoInexistente )
            {
                System.err.println(" Arquivo inexistente ou arquivo não pode ser criado");
                System.exit(1);
            }
        }
```

Manipulação de arquivos

```
        public void escrever()
        {
            try
            {
                arquivo.format("Escrita no arquivo realizada com sucesso");
            }
            catch(FormatterClosedException formatoDesconhecido)
            {
                        System.err.println("Erro ao escrever");
                        return;
            }
            catch(NoSuchElementException excecaoElemento)
            {
                System.err.println("Entrada invalida. Por exemplo, era pra ser uma string, mas foi um inteiro");
            }

        }

       public void fechar()
       {
           arquivo.close();
       }
}
```

Manipulação de arquivos

Escrita de um arquivo

• Para criarmos um arquivo , iremos utilizar a classe FileOutputStream que irá receber um String com o nome do arquivo a ser gravado. Caso já existir um arquivo com mesmo nome, este será sobreposto , ou seja , as informações serão apagadas

Leitura de uma arquivo

• Para abrir este arquivo, iremos usar a classe FileInputStream que irá receber uma String que será o nome do arquivo a ser aberto

```
public class EscreverLerArquivo {
public void escreverArquivo() {
        FileOutputStream arq;
        try {
                arq = new FileOutputStream(“arquivo.txt");
                 arq.write(97);
                 arq.close();
                System.out.println("Arquivo Gravado");
        } catch (FileNotFoundException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
    }
}
```

Manipulação de arquivos

```
public class TestaArquivo {
public static void main(String[] args) {
EscreverLerArquivos arq = new
EscreverLerArquivos();
arq.escreverArquivo();
 try {
arq.lerArquivo();
} catch (FileNotFoundException e) {
System.out.println("Arquivo não
encontrado"); }catch(IOException e) {
 System.out.println("Arquivo
corrompido");
} } }
```

```
public void lerArquivos()throws IOException {
FileInputStream arq = new
FileInputStream("arquivos.txt");
InputStreamReader entrada = new
InputStreamReader(arq);
int c = entrada.read(); // método read devolve
um int
System.out.println(c);
}
```

Resultado: 97

Manipulação de arquivos

- Para podermos exibir na tela o caractere 'a', basta fazermos o cast para char:System.out.print((char) c);
- Abra o arquivo file e escreva uma frase
- Modifique o programa só no sysout :
- while(c!=-1) {
- System.out.print((char) c);
- c = entradaFormatada.read();
- }

Manipulação de arquivos

Lendo uma String

• A classe BufferedReader que recebe como argumento um objeto do tipo InputStreamReader e agrupa os caracteres até formar uma linha

```
public class LendoArquivoBuffered {
public static void main(String[] args) throws IOException {
•            FileInputStream entrada;
•   try {
•   arq = new FileInputStream("file.txt");
•   InputStreamReader entrada = new InputStreamReader(arq);
•    BufferedReader entradaString = new BufferedReader(entrada);
•    String linha = entradaString.readLine(); // lê string
•     while(linha != null) {  // quando chega no final da ultima linha retorna null.
•    System.out.println(linha+"\n");
•    linha = entradaString.readLine();
•     }
•   } catch (FileNotFoundException e) {
•   e.printStackTrace();
•   } catch (IOException e) {
•   e.printStackTrace();
•   }
```

Manipulação de arquivos

Fechando arquivos - O método close()

• Independente da complexidade de seus aplicativos Java é importante você fechar os arquivos que abriu (seja pra ler ou escrever). Isso é feito através do método **close()**, presentes nas classes que usamos para tratar os arquivos.

Acrescente a linha depois do while :

entrada.close();

Serialização de objetos

• Java permite escrever objetos inteiros em arquivos .

• Trabalharemos com fluxo de dados (arquivos binários) .

• A classe deve implementar a interface **Serializable**

Manipulação de arquivos

Serialização de objetos

• Serialização é um mecanismo para ler e gravar um objeto inteiro a partir de um arquivo.

• Um objeto serializado é um objeto representado como uma sequência de byte que inclui:

– Dados do objeto

– Informações sobre o tipo do objeto

– Tipos de dados armazenados do objeto

• O objeto pode ser desserializado a partir do arquivo gravado.

Manipulação de arquivos

- As classes ObjectInputStream e ObjectOutputStream permitem que objetos sejam lidos ou gravado em fluxo.

• Para usar a serialização com arquivos inicializaremos esses objetos de fluxo com objetos de fluxo que leem e gravam bytes em arquivos:
FileInputStream e **FileOutputStream**

Abertura de fluxo para leitura
FileInputStream objeto = new **FileInputStream**(“objeto.bin ”);
ObjectInputStream entrada = new **ObjectInputStream** (objeto);

Abertura de fluxo para escrita
FileOutputStream objeto = new **FileOutputStream** (“objeto.bin ”);
ObjectOutputStream saida= new **ObjectOutputStream** (objeto);

Manipulação de arquivos

- Leitura de dados :
- objeto= (Tipo) entrada.**readObject();**
- Escrita de dados :
- saida.**writeObject(objeto);**
- Fechamento do arquivo :
- entrada.**close();**
- saida.**close();**

```
public class Crianca implements Serializable{
private String nome;
private int idade;
public Crianca(String nome, int idade) {
super();
this.nome = nome;
this.idade = idade;
}
} Crie os Getters
```

Manipulação de arquivos

- A interface Serializable é uma interface de marcação, não tem nenhum método.
- A classe é marcada para permitir que os objetos sejam serializaveis.
- Devemos verificar que cada atributo da classe também seja serializável.Caso tivéssemos como atributo Objeto de outra classe esta também teria que ser serializada

```
public class EscreverLerObjetos {
public  void escreverObjeto(Crianca crianca) {
FileOutputStream fluxo;
try {
fluxo = new FileOutputStream("Crianca.bin");
ObjectOutputStream objeto = new ObjectOutputStream(fluxo);
objeto.writeObject(crianca);
objeto.close();
System.out.println("Arquivo Gravado");
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
} }
```