LCP - Aula 05

Prof. Lucas Guerreiro

Aula anterior

- Tipo primitivo char
- Conceitos de Strings
- Manipulação de Strings
- Expressões Regulares
- Exercícios

Aula 05

Objetivo: Exceções em Java. Tratamento de Exceções. Manipulação de Arquivos.

Exceções

- Exceções são "problemas" em tempo de execução.
- A tendência é que exceções sejam casos isolados, ou seja, ocorram com pouca frequência.
- O que acontece se tentarmos passar um valor float para uma variável que espera um valor inteiro?

```
int numero = scan.nextInt(); // no console inserir 4.5
```

```
int[] lista = new int[10];

lista[10] = 5;
```

Tratamento de exceções

- Exceções não tratadas geram erros em tempo de execução e interrompem o programa.
- Solução: tratamento de exceções
- Lidar com exceções esperadas e apresentar um caminho alternativo ao invés do "crash" no programa
- Com tratamento de exceções o programa continua sua execução e se torna mais tolerante a falhas.
- Conseguimos identificar diferentes tipos de exceções e aplicar diferentes tratamentos a cada um deles.

Try/Catch

- ➔ Podemos identificar blocos passíveis de exceções e deixá-los dentro de uma instrução `try`.
- ➔ A instrução `catch` irá lidar com a exceção esperada.
- ➔ Podemos ter múltiplas instruções `catch` para cada `try`.
- ➔ Ao se executar um código dentro do `try` que não gera exceção, o bloco `catch` não é executado.
- ➔ `finally`: comando opcional para comandos serem executados após as verificações de `try`/`catch`, com isso podemos tomar ações depois das checagens de exceção ou a ocorrência das exceções do bloco. Ou seja, independente do que ocorre no `try`/`catch`, temos a garantia de que o bloco `finally` será executado.

Exemplo

Notas a serem atribuídos a "alunos"

```
String[] nomes = {"Aluno1", "Aluno2", "Aluno3"};
int[] notas_alunos = new int[3];
int[] notas = {4, 7, 8, 9};

for (int i = 0; i < notas.length; i++){
    notas_alunos[i] = notas[i];
}

System.out.printf("%n%s%10s", "Aluno", "Nota");
System.out.printf("%n---------------");

for (int i = 0; i < nomes.length; i++){
    System.out.printf("%n%5s%8d", nomes[i], notas_alunos[i]);
}
```

Exceções0.java

Exemplo

```
String[] nomes = {"Aluno1", "Aluno2", "Aluno3"};
int[] notas_alunos = new int[3];
int[] notas = {4, 7, 8, 9};

for (int i = 0; i < notas.length; i++){
     try{
          notas_alunos[i] = notas[i];
     }
     catch (ArrayIndexOutOfBoundsException e){
          System.out.println("Nota excedente. Lista de alunos menor do que lista de
     notas.");
          System.out.println("Verificar nota: " + notas[i]);
          System.out.println("Erro: " + e);
     }
}
System.out.printf("%n%s%10s", "Aluno", "Nota");
System.out.printf("%n---------------");

for (int i = 0; i < nomes.length; i++){
     System.out.printf("%n%5s%8d", nomes[i], notas_alunos[i]);
}
```

Exceções1.java

Exercício

Adaptar o modelo anterior, de forma que recebamos do usuário (via console) as notas, e estas deverão ser notas inteiras. Em caso de receber um tipo diferente (exceção ao alimentar variável `int`), exibir mensagem e solicitar nova nota que seja do tipo `int` (com uso de `try`/`catch`).

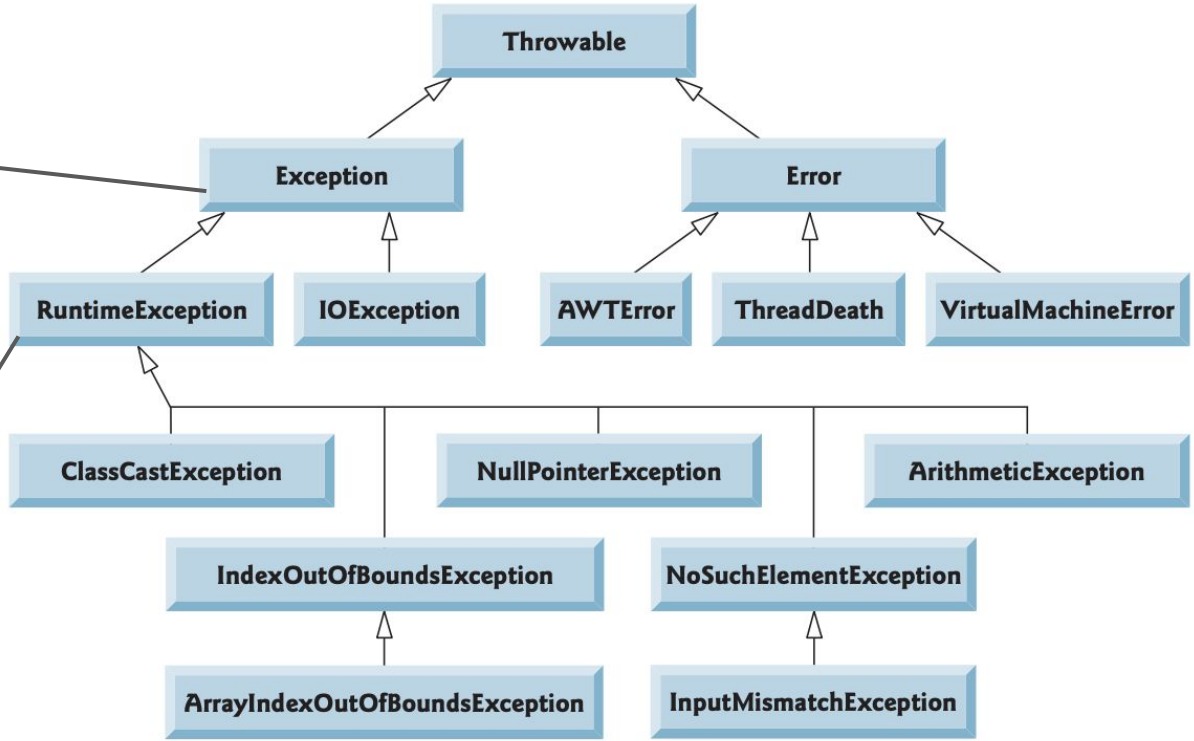
```
Scanner scan = new Scanner(System.in);

int[] notas = {-1, -1, -1};

for (int i = 0; i < notas.length; i++) {
    while (notas[i] == -1) {
        System.out.printf("%nDigite a %dª nota: ", i+1);
        try {
            notas[i] = Integer.parseInt(scan.nextLine());
        } catch (InputMismatchException e) {
            System.out.print("Atenção! Nota inválida!");
        } catch (NumberFormatException e) {
            System.out.print("Atenção! Nota inválida!");
        }
    }
}

for (int i = 0; i < notas.length; i++) {
    System.out.printf("%nNota %d: %d", i+1, notas[i]);
}

scan.close();
```

Exceções2.java

Hierarquia de Exceções

Exceções verificadas: ocasionadas por situações além do controle do programa. Devem ser tratadas para compilar. Exemplo: tentar abrir arquivo que não existe.

Exceções não verificadas: em geral são ocasionadas por erros da arquitetura do código. Não obrigatoriamente devem ser tratadas. Exemplo: divisão por zero.



Lançando exceções

- Métodos (e trechos de código) podem lançar exceções para serem capturadas por quem está chamando o método.
- Ao utilizar `método() throws Exceção` na assinatura de um método, o código que chama o método deve capturar exceções do tipo Exceção.
- Podemos ainda identificar determinado trecho de código que deve lançar uma exceção, ou seja, indicar um problema de acordo com determinada condição. Nestes casos, utilizamos a cláusula `throw` para lançar a exceção.

```
public static int divisao(int numerador, int denominador) throws ArithmeticException {
    return numerador / denominador;
}

public static void main(String[] args){
    Scanner scanner = new Scanner(System.in);
    boolean continuarLoop = true;

    do{
        try{
            System.out.print("Digite o numerador (inteiro): ");
            int numerador = scanner.nextInt();
            System.out.print("Digite o denominador (inteiro): ");
            int denominador = scanner.nextInt();
            int resultado = divisao(numerador, denominador);
            System.out.printf("%nResultado: %d / %d = %d%n", numerador, denominador, resultado);
            continuarLoop = false;
        }catch (InputMismatchException inputMismatchException) {
            System.err.printf("%nExceção: %s%n", inputMismatchException);
            scanner.nextLine();
            System.out.printf("Você deve inserir um número inteiro. Tente novamente.%n%n");
        }catch (ArithmeticException arithmeticException) {
            System.err.printf("%nExceção: %s%n", arithmeticException);
            System.out.printf("Denominador deve ser diferente de zero. Tente novamente.%n%n");
        }
    } while (continuarLoop);
}
```

Exceções3.java

Exemplo

Tratar o caso de notas inválidas, sendo que as notas devem estar entre 0 e 10. Caso contrário, uma exceção deve ser lançada.

Exemplo

```
static int leNota(int i) {// throws IllegalArgumentException{
	Scanner scan = new Scanner(System.in);
	System.out.printf("%nDigite a %dª nota: ", i+1);
	int nota = scan.nextInt();// Integer.parseInt(scan.nextLine());
	scan.nextLine();
	if (nota < 0 || nota > 10) {
		throw new IllegalArgumentException("Nota deve ser entre 0 e 10.%n");
	}
	return nota;
}

public static void main(String[] args){

	int[] notas = {-1, -1, -1};

	for (int i = 0; i < notas.length; i++) {
		while (notas[i] == -1) {

			try {
				notas[i] = leNota(i);
			} catch (InputMismatchException e) {
				System.out.print("Atenção! Nota inválida! Deve ser um número inteiro. Erro = " + e + "\n");
			} catch (IllegalArgumentException e) {
				System.out.print("Atenção! Nota inválida! Deve ser um número inteiro entre 0 e 10. Erro = " + e
+ "\n");
			} finally {
				System.out.print("Nota lida.\n");
			}
		}
		System.out.print("Fim de leitura da nota.");
	}

	for (int i = 0; i < notas.length; i++) {
			System.out.printf("%nNota %d: %d", i+1, notas[i]);
	}

}
```

Exceções4.java

Arquivos

- Fluxo de bytes sequencial.
- Fluxos de arquivos podem ser:
  - baseados em bytes - manipulam dados em formato binário (arquivos binários);
  - baseados em caracteres - manipulam dados como sequências de caracteres (arquivos de texto).

```
Scanner arquivo = new Scanner(new FileReader("arquivo.txt",
StandardCharsets.UTF_8));
while (arquivo.hasNextLine()) {
    String linha = arquivo.nextLine();
    System.out.println(linha);
}

String texto = Files.readString(Path.of("arquivo.txt"));

FileInputStream inputStream = new FileInputStream("arquivo.txt"); //bytes
```

Dicas: https://www.baeldung.com/reading-file-in-java
https://www.geeksforgeeks.org/different-ways-reading-text-file-java/

Arquivos0.java

Arquivos

```
Scanner arquivo = null;
try {
    arquivo = new Scanner(new FileReader(raiz + "arquivo.txt",
StandardCharsets.UTF_8));
} catch (FileNotFoundException e) {
    System.err.println("Arquivo não existe. Erro " + e);
    System.exit(1);
} catch (IOException e) {
    e.printStackTrace();
    System.exit(1);
} finally {
    while (arquivo.hasNextLine()) {
        String linha = arquivo.nextLine();
        System.out.println(linha);
    }
    arquivo.close();
}
```

Arquivos1.java

Ler arquivo com nomes

```
try {
     arquivo = new Scanner(new File(raiz + "dados.txt"));
}catch (FileNotFoundException e) {
     System.err.println("Erro: arquivo não encontrado. Encerrando...");
     System.exit(1);
} finally {
     int contador = 0;
     while (arquivo.hasNext()) {
          contador++;
          String linha = arquivo.nextLine();
          String[] nome = linha.split(" ");
          String primeiroNome = nome[0];
          String sobrenome = nome[1];
          System.out.printf("Pessoa %d. Nome: %s, sobrenome: %s%n", contador,
primeiroNome, sobrenome);
     }
}
```

Arquivos2.java

Ler tipos diferentes de dados com Scanner

```
try {
	arquivo = new Scanner(new File(raiz + "dados2.txt"));

	int contador = 0;
	while (arquivo.hasNext()) {
		contador++;
		String linha = arquivo.nextLine();

		Scanner dadosLinha = new Scanner(linha).useDelimiter("\\s/\\s");

		String nome = dadosLinha.next();
		String curso = dadosLinha.next();
		int idade = dadosLinha.nextInt();
		System.out.printf("Aluno %d. Nome: %s, curso: %s, idade: %d%n", contador, nome, curso, idade);

	}
}catch (FileNotFoundException e) {
	System.err.println("Erro: arquivo não encontrado. Encerrando...");
	System.exit(1);
} finally {
	arquivo.close();
}
```

Arquivos3.java

Exercício

A partir de um arquivo que contenha dados de funcionários, no formato:

```
Nome; Cargo; Salário
```

alimentar uma lista de objetos funcionários (com atributos nome, cargo e salário) com os respectivos dados.

Ao fim, exibir cada um dos funcionários da lista com todos seus dados (toString).

Exercício

```
private String nome;
private String cargo;
private double salario;

public FuncionariosArquivo(String nome, String cargo, double salario) {
     this.nome = nome;
     this.cargo = cargo;
     this.salario = salario;
}

@Override
public String toString() {
     return String.format("Funcionário: %s, cargo: %s, salário: %.2f", nome, cargo,
salario);
}
```

FuncionariosArquivo.java

```
public static void main(String[] args) {

	List<FuncionariosArquivo> funcionarios = new ArrayList<>();

	String raiz = "D://Users//Lucas//";
	Scanner arquivo = null;

	try {
		arquivo = new Scanner(new File(raiz + "dados3.txt"));
		while (arquivo.hasNext()) {
			String linha = arquivo.nextLine();
			Scanner dadosLinha = new Scanner(linha).useDelimiter("\\s/\\s");
			String nome = dadosLinha.next();
			String cargo = dadosLinha.next();
			double salario = Double.parseDouble(dadosLinha.next()) ;

			funcionarios.add(new FuncionariosArquivo(nome, cargo, salario));

		}
	}catch (FileNotFoundException e) {
		System.err.println("Erro: arquivo não encontrado. Encerrando...");
		System.exit(1);
	} finally {
		arquivo.close();
		System.out.println("---------- FUNCIONARIOS ------------");
		for (FuncionariosArquivo funcionario : funcionarios) {
			System.out.println(funcionario);
		}
	}
}
```

FuncionariosArquivo.java

Manipulação de Fluxo

Até agora fizemos apenas leituras de arquivos com a classe FileReader

- método `read();`

Podemos fazer uso da classe FileWriter para escrever em arquivos

- método `write();`

Cópia de arquivo

```
FileReader entrada;
FileWriter saida;
try {
    entrada = new FileReader(raiz + "entrada.txt");
    saida = new FileWriter(raiz + "saida.txt");
    int c;
    while ((c = entrada.read()) != -1) {
        saida.write(c);
    }
    entrada.close();
    saida.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
    System.exit(1);
} catch (IOException e) {
    e.printStackTrace();
    System.exit(1);
} finally {
    System.out.println("Cópia terminada.");
}
```

Arquivos4.java // Arquivos5.java

Exercício

Criar arquivo com números de 1 a 10 aleatoriamente ordenados e uma string associada a cada número, gerar um novo arquivo com a numeração ordenada.

Exemplo:

3 ABC

1 GHI

2 ZZZ

Novo arquivo:

1 GHI

2 ZZZ

3 ABC

Exercício

Tratar um arquivo no formato Nome / Idade e gerar um novo arquivo com somente as entradas de pessoas que tenham nome iniciado em A .. L ou que tenham idade entre 20 e 30 anos.

Exemplo na entrada:

Alberto / 15

Maria / 35

Fulano / 25

Teríamos no novo arquivo somente:

Alberto / 15

Fulano / 25