Aula 7

Tratamento de erros

No andamento da execução de programas podem ocorrer erros imprevistos:

- Esses erros são conhecidos como exceções;
- Podem ser provenientes de:
  - Erros de lógica
  - Acesso a dispositivos
  - Arquivos externos.

Tratamento de erros

- Erros de lógica
  - Tentar manipular um objeto que está com o valor nulo.
  - Dividir um número por zero.
  - Tentar manipular um tipo de dado como se fosse outro.
  - Tentar utilizar um método ou classe não existentes.
- Arquivos externos.
  - Tentar abrir um arquivo que não existe.
  - Tentar fazer consulta a um banco de dados que não está disponível.
  - Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita.
  - Tentar conectar em servidor inexistente.

Tratamento de erros

Qual a importância de trabalhar com tratamento de erros?

- Para que o sistema não quebre a sua execução;
- Quando a execução é quebrada o sistema para de funcionar;
- Isso gera inconsistência, insatisfação e falta de confiança na operabilidade do sistema;
- Um sistema com controle de erros não se perde na execução;
  - Permite que seja feita uma ação para contorna-lo;
  - Permite que seja gravado um log do erro para posterior debug por parte do desenvolvedor;
  - Permite que se possa informar ao usuário sobre o ocorrido e o que fazer;

Tratamento de erros

Para tratar exceções em Java são utilizados os comandos try e catch.

```
try {
    DPRelatorio dpr = new DPRelatorio();
    DPFuncionario funcionario = null;

    DPDiretor diretor = new DPDiretor();
    diretor.setNome("Marcelo");
    diretor.setCpf("123.456.798.-10");
    diretor.setSalario(2000.00);
    System.out.println(dpr.calculaFolha(funcionario));
catch (Exception e) {
//Tratamento da exceção
}
```

Arrays

Vamos aprender hoje:

- Array
- Collections Framework
  - List
  - ArrayList
  - Generics
  - Set
- Map

Arrays

Podemos declarar diversas variáveis e usá-las:

```
double var1 = 1.0;
double var2 = 2.0;
double var3 = 3.0;
double var4 = 4.0;
```

Arrays

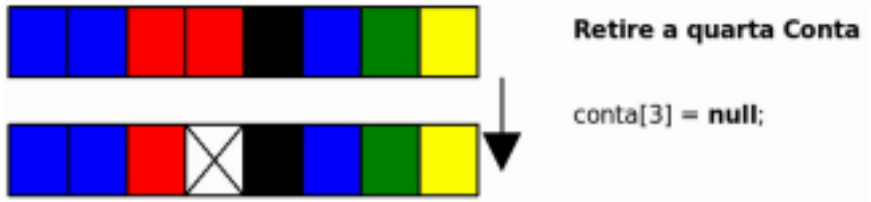
Podemos declarar um vetor (array) de double;

```
int tamanho = 10;
double[] arrayVar;
arrayVar = new double[tamanho];

for(int i = 0; i < tamanho; i++)
	arrayVar[i] = i * tamanho;

for(int i = 0; i < arrayVar.length; i++)
	System.out.println(arrayVar[i]);
```

Array

Manipulação de arrays

- Manipular é bastante trabalhoso.
- Essa dificuldade aparece em diversos momentos:
  - Não podemos redimensionar um array em Java;
  - É impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
  - Não conseguimos saber quantas posições do array já foram populadas sem criar, para isso, métodos auxiliares



Collections

- Foi inserida no Java na versão 2;
- A API de Collections é robusta;
- Diversas classes com estruturas de dados avançadas.
- Oferece diversas estruturas para utilização:
  - List
  - ArrayList
  - Generics
  - Set

Collections - List

List

- É uma coleção que permite elementos duplicados ;
- Mantém uma ordenação específica entre os elementos;
- Trabalha com um array interno para gerar uma lista;
- A implementação mais utilizada da interface List é a ArrayList;
- Sim, List é uma interface;

Collections - ArrayList

ArrayList

```
List lista = new ArrayList();
lista.add("Manoel");
lista.add("Joaquim");
lista.add(“Maria");
lista.remove(1);
lista.size();
```

Collections - ArrayList

ArrayList

- ArrayList não é um array;
- Internamente usa um array como estrutura para armazenar os dados;
- Porém este atributo está propriamente encapsulado;
- Não podemos acessá-lo,
- Não podemos usar [ ]
- Não podemos acessar o atributo length;

Collections - ArrayList

ArrayList

- Operacionais - add(), remove(), clear(), get();
- Informativos - size(), isEmpty(), sort();
- Uma lista é uma excelente alternativa a um array comum;
- Todos os benefícios de arrays, sem cuidado com remoções e espaço;
- Trabalha do modo mais genérico possível;
- Não há uma ArrayList específica para Strings , outra para Números, outra para Datas etc;
- Todos os métodos trabalham com Object;

Collections - ArrayList

ArrayList

- Isso mesmo! Em uma lista, é possível colocar qualquer Object .
- Com isso, é possível misturar objetos:
- E na hora de recuperar esses objetos?
- Como o método get devolve um Object. Precisamos fazer o cast.
- Mas com uma lista com vários objetos de tipos diferentes, isso pode não ser tão simples...
- Geralmente usamos listas como mesmo tipo de dado.

Collections - ArrayList

ArrayList

```
Funcionario func = new Funcionario()
ContaCorrente cc = new ContaCorrente()
List lista = new ArrayList();
lista.add("Manoel");
lista.add("Joaquim");
lista.add(func);
lista.add(cc);
Funcionario func2 = (Funcionario) lista.get(3);
lista.size();
```

Generics

- No Java 5.0, foi adicionado o recurso para restringir as listas a um determinado tipo de objetos (e não qualquer Object);

```
List<ContaCorrente> contas = new ArrayList<ContaCorrente>();
contas.add(c1);
contas.add(c3);
contas.add(c2);
contas.add("uma string qualquer"); // isso não compila mais!!
```

- O uso de Generics também elimina a necessidade de casting

Collections - Generics

Generics

- A partir do Java 7 houve uma simplificação na sintaxe do generics;
- Se instanciarmos na mesma linha da declaração, não é necessário passar os tipos novamente, basta usar new ArrayList<>() .
- É conhecido como operador diamante:

```
List<ContaCorrente> contas = new ArrayList<>();
```

Ordenação de Listas

- A classe Collections traz um método estático sort que recebe um List como argumento e o ordena por ordem crescente. Por exemplo:

```
List<String> lista = new ArrayList<>();
lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");
// repare que o toString de ArrayList foi sobrescrito:
System.out.println(lista);
Collections.sort(lista);
System.out.println(lista);
```

Ordenação de Listas

- Para ordenar objetos precisamos determinar um critério;
- Esse critério irá determinar qual elemento vem antes de qual.
- É necessário instruir o sort sobre como comparar um objeto
- Para isto, o método sort necessita que todos os objetos da lista sejam **comparáveis**;
- Isso se dará através de um método que fará tal comparação com outro objeto;
- Como é que o método sort terá a garantia de que a sua classe possui esse método?
- Isso será feito, novamente, através de um contrato, de uma interface!