



PROGRAMAÇÃO ORIENTADA A OBJETOS [POO]

Material 08 – POO_08

Prof. Mestre Marcos Roberto de Moraes [Maromo]

Tratamento de Exceções (Como gerenciar os erros em Java)

Agenda

- ❑ Introdução ao Tratamento de Erros
- ❑ Estrutura do Tratamento
- ❑ Criando Exceções Customizadas
- ❑ Proteção do Código



Introdução

- ❑ Exceção é alguma condição excepcional ou não esperada.
 - ▣ Em aplicativos Java, uma exceção é disparada quando o programa se comporta de uma forma que deve ser tratada;
 - ▣ Quando o Programa ou o SO gera erros ou
 - ▣ Até mesmo quando uma chamada de métodos resulta valores inesperados.

Erros

❑ Erros de lógica.

- ▣ Estes erros se apresentam no desenvolvimento de um algoritmo não apropriado para solucionar o problema que se propõe. Estes erros não necessariamente causam interrupção na execução do programa.

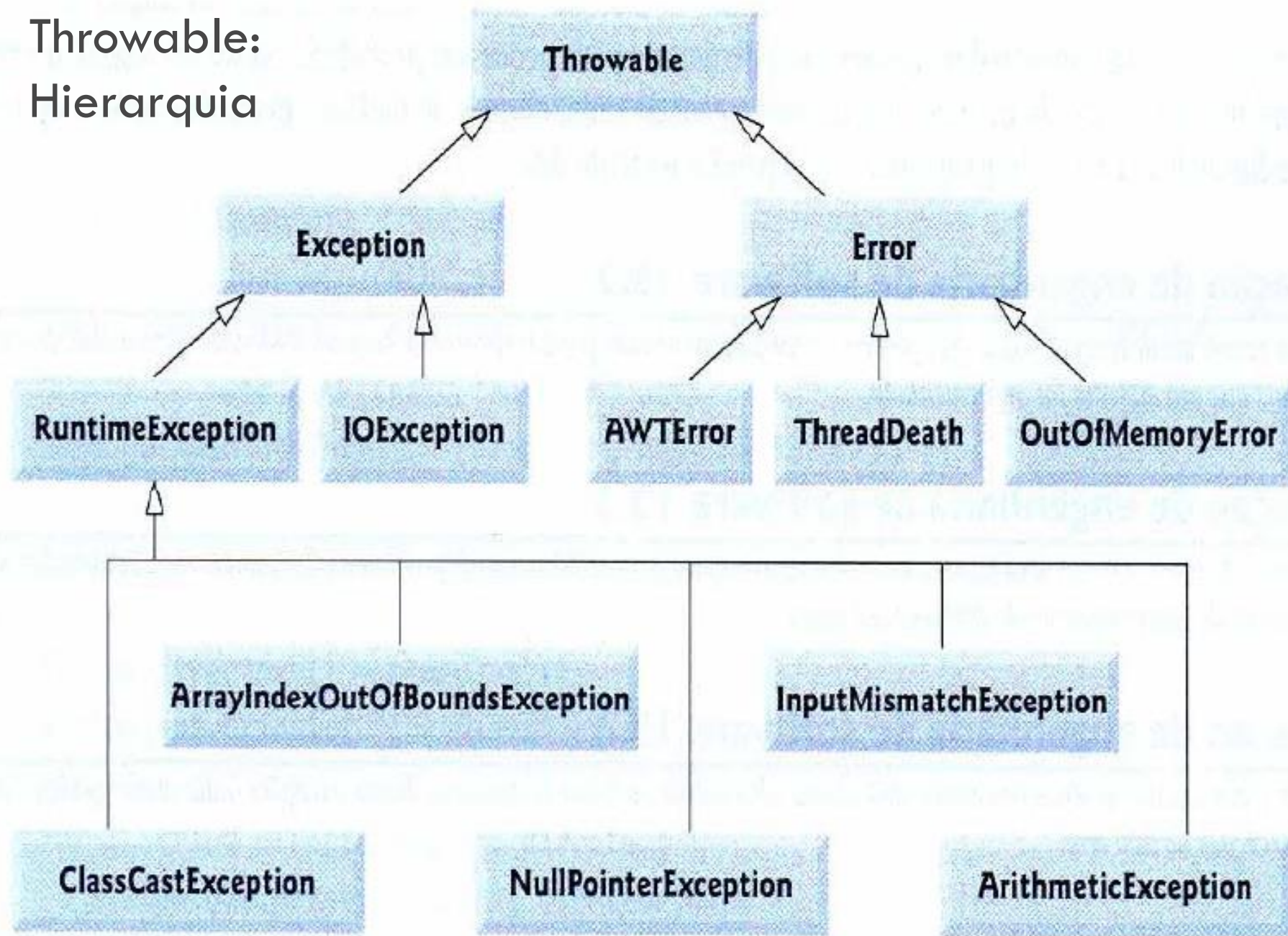
❑ Erros de execução.

- ▣ São erros mais específicos em relação aos lógicos, que decorrem de uma operação inválida e causam interrupção na execução do programa.
- ▣ Ao se causar um erro a execução é interrompida e é enviado um sinal ao programa indicando que a operação não pode ser realizada.

Throwable

- ❑ As linguagens de programação possuem formas para identificar e tratar os erros de execução.
- ❑ Em Java eles são detectados pela **JVM** e **um objeto de uma classe que caracteriza o erro é criada**.
- ❑ Notifica-se o programa da operação inválida e caso seja possível tratá-lo, pode-se acessar o objeto que caracteriza o erro.
- ❑ Os erros são caracterizados por objetos de classes específicas que pertencem a hierarquia da classe **Throwable**.

Throwable: Hierarquia



Throwable

- ❑ A classe **Throwable** é a superclasse da classe **Exception** e, portanto, também é a **superclasse de todas as exceções**.
- ❑ Somente objetos Throwable podem ser utilizados como mecanismo de tratamento de exceções.
- ❑ A classe Throwable tem duas subclasses:
 - ❑ **Exceptions**
 - ❑ **Error**

Classe Error

- ❑ A classe Error, com suas subclasses, é **utilizada para indicar erros graves que não se espera que sejam tratados pelo programa.**
- ❑ **Errors raramente acontecem** e não são de responsabilidade da aplicação.
- ❑ Exemplos de Errors:
 - ▣ Erros internos da JVM.
 - ▣ Falta de memória.

Classe Exception

- ❑ Normalmente chamamos os erros em Java de **exceptions**.
- ❑ Uma exceção representa uma situação que normalmente não ocorre (ou não deveria ocorrer) e representa algo de estranho ou inesperado no sistema.
- ❑ O Java distingue entre duas categorias de exceções:
 - ▣ Checked (verificadas)
 - ▣ Unchecked (não verificadas)

Unchecked Exceptions

- ❑ Uma exceção não verificada é aquela em que o compilador Java **não verifica o código para determinar** se ela foi capturada ou declarada.
- ❑ Em outras palavras, o programador não é obrigado a inserir o tratamento de erro.
- ❑ Em geral, o programador pode impedir a ocorrência de exceções não verificadas pela codificação adequada.

Unchecked Exceptions

- ❑ Todos os tipos de exceção que são subclasses diretas ou indiretas da classe **RuntimeException** são exceções não verificadas.
- ❑ Exemplos de Unchecked Exceptions:
 - ▣ Entrada de tipos incompatíveis (Leitura de uma String em um atributo double, por exemplo)
 - ▣ Acesso a índice inexistente em um array.
 - ▣ Chamada a um método de um objeto nulo.

Checked Exceptions

- ❑ Uma **exceção verificada** (ao contrário da não verificada) é aquela em que o compilador Java verifica o código para determinar se ela foi capturada ou declarada e obriga o programador a inserir um tratamento de erro.
- ❑ Todos os tipos de exceção que herdam da classe `Exception`, mas não da `RuntimeException`, são exceções verificadas.
- ❑ Exemplos de Checked Exceptions:
 - ▣ Abrir um arquivo para leitura (onde pode ocorrer o erro do arquivo não existir).

Tratamentos de Checked Exceptions

- Existem duas formas de tratar uma Checked Exception:
 - Utilizando a cláusula **throws**
 - Utilizando a estrutura **try-catch-finally**

Throws

- ❑ O **Throws** delega para quem chamou a responsabilidade de tratar o erro, ou seja, a obrigatoriedade de tratamento é passada para a classe que fará a chamada ao método.

Blocos:

try – catch – finally

- ❑ É a principal estrutura para captura de erros em Java onde o código irá tentar (**try**) executar o bloco “perigoso” e, caso ocorra algum problema, o erro gerado será pego (caught).
- ❑ Um **catch** possui um parâmetro de exceção (identificação do erro) seguido por um bloco de código que o captura e possibilita o tratamento.
- ❑ É possível definir vários catch para cada try.
- ❑ O **finally** é opcional e, se for usado, é colocado depois do último catch.
- ❑ Havendo ou não uma exception (identificada no bloco try) o bloco finally sempre será executado.

Blocos:

try – catch – finally

```
public class SampleTryTest {  
    public static void main(String[] args) {  
        try {  
            //Bloco perigoso [que pode gerar o erro]  
        } catch (Exception e) {  
            //Tratamento da exceção [Podem ser vários catches]  
        } finally {  
            //Bloco que será sempre executado [Opcional]  
        }  
    }  
}
```


Exemplos e exceptions

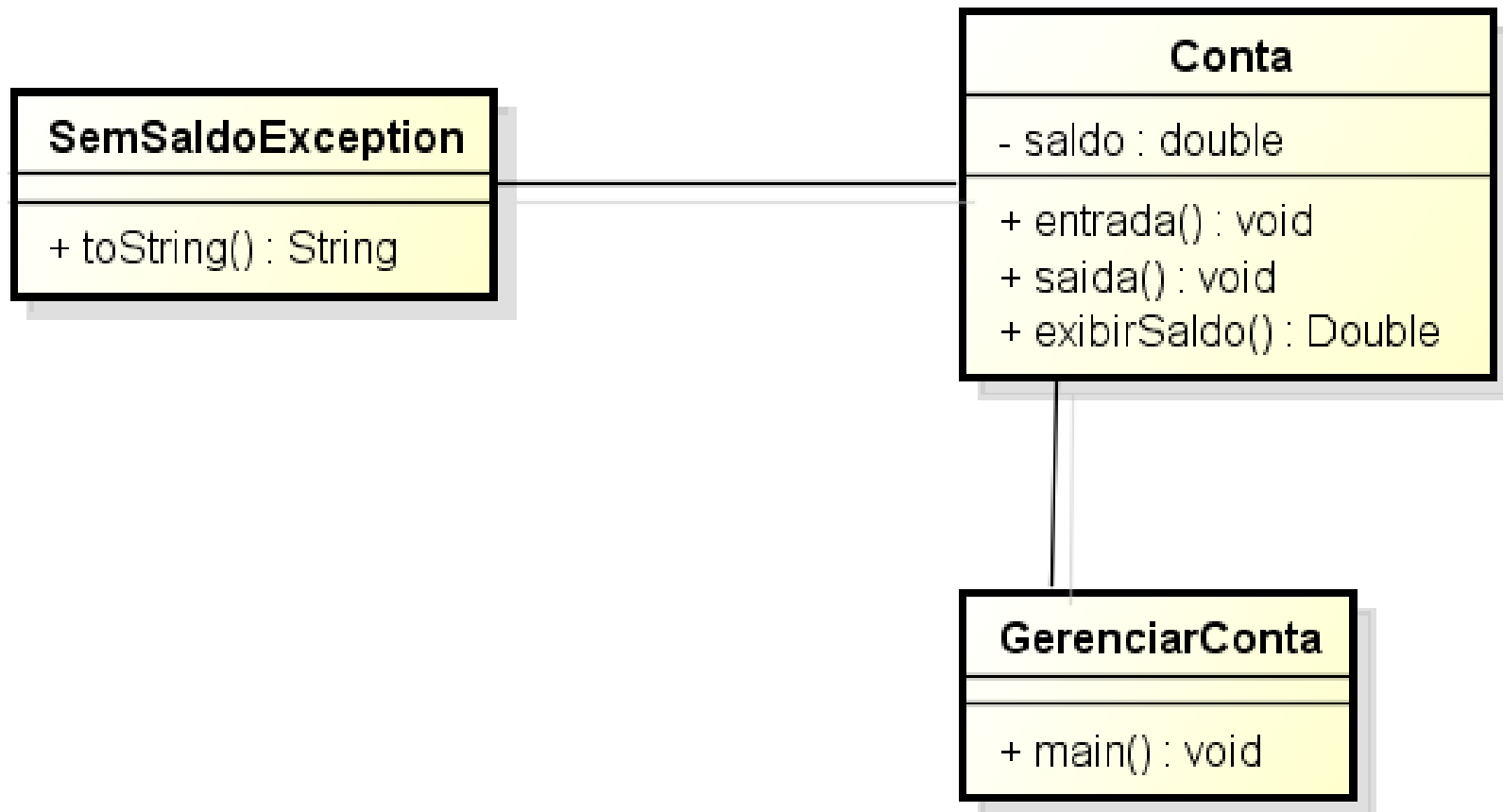
- ❑ `ArithmeticException`
 - ▣ Resultado de uma operação matemática inválida.
- ❑ `NullPointerException`
 - ▣ Tentativa de acessar um objeto ou método antes do mesmo ser instanciado.
- ❑ `ArrayIndexOutOfBoundsException`
 - ▣ Tentativa de acessar um elemento de um vetor além de sua dimensão (tamanho) original.
- ❑ `NumberFormatException`
 - ▣ Incompatibilidade de tipos numéricos.
- ❑ `FileNotFoundException`
 - ▣ Arquivo não encontrado.
- ❑ `ClassCastException`
 - ▣ Tentativa de conversão incompatível.

Exception

□ Exception.

- ▣ É possível capturar o erro recorrendo a classe mãe das exceções, ou seja, qualquer erro será interceptado.
- ▣ Esse tipo de tratamento é desaconselhável em função da falta de especificação do erro ocorrido.

Aplicação: ProjContaTry



Classe:

SemSaldoException

Exceção Personalizada

- ❑ O programador também pode criar suas próprias classes de erros.
- ❑ Para ser reconhecida pelo compilador Java como uma classe de erro ela deve pertencer a hierarquia de Throwable, mais especificamente de Exception.
- ❑ Lembrando que toda subclasse de Exception que não for subclasse de RuntimeException é uma Checked Exception, ou seja, seu tratamento é obrigatório.

Classe:

SemSaldoException

```
package prjcontatry;
```

```
public class SemSaldoException extends Exception {
```

```
    @Override
```

```
    public String toString() {
```

```
        return "Não possui saldo suficiente para o saque";
```

```
    }
```

```
}
```

Classe:

SemSaldoException

```
public void saida() throws SemSaldoException {
    try {
        double valor;
        valor = Double.parseDouble(JOptionPane.showInputDialog(null, "Valor da Retirada"));
        if (valor > this.saldo) {
            throw new SemSaldoException();
        } else {
            this.saldo -= valor;
            JOptionPane.showMessageDialog(null, "Saldo R$ " + this.saldo);
        }
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage());
    } catch (SemSaldoException e1) {
        JOptionPane.showMessageDialog(null, "Erro: " + e1.toString());
    }
}
```

Classe:

SemSaldoException

```
public double exhibirSaldo() {  
    return this.saldo;  
}  
  
public void entrada() {  
    try {  
        double valor;  
        valor = Double.parseDouble(JOptionPane.showInputDialog(null, "Valor do Depósito"));  
        this.saldo += valor;  
    } catch (NumberFormatException e) {  
        JOptionPane.showMessageDialog(null, "Erro: " + e.getMessage());  
    }  
}
```

GerenciarConta

```
package prjcontatry;

import javax.swing.JOptionPane;

public class GerenciarConta {

    public static void main(String[] args) throws SemSaldoException{
        Conta c1 = new Conta();
        c1.entrada();
        double valor = c1.exibirSaldo();
        JOptionPane.showMessageDialog(null, "Saldo: " + valor);
        c1.saida();
        valor = c1.exibirSaldo();
        JOptionPane.showMessageDialog(null, "Saldo: " + valor);
    }
}
```


Desafio

Missão

- ❑ Crie um calculadora, utilizando os conhecimentos de Swing e de Tratamento de Erros. O Programa deve possuir as classes:
 - ❑ IniciarCalculadora (contém o método main)
 - ❑ JfCalculadora (Jframe com a interface gráfica da calculadora, vide figura)
 - ❑ Calculadora (classe de modelagem, com os seguintes atributos: valor (double) e operação (int); deve possuir os métodos: somar, subtrair, dividir e multiplicar.

Exemplo GUI Calculadora



Referências

□ Bibliográficas:

- ▣ Mendes – Java com Ênfase em Orientação a Objetos [Exercícios do Capítulo 1]
- ▣ Brasport – Certificação Java 6 – A Bíblia – Serson, R.R.
- ▣ Deitel – Java, como programar – 6º edição.
- ▣ Arnold, Gosling, Holmes – A linguagem de programação Java – 4º edição.
- ▣ Apostilas Caelum
- ▣ Material do Curso de Capacitação Java do CPS

□ Internet

- ▣ <http://java.sun.com>
- ▣ <http://www.guj.com.br>
- ▣ <http://www.portaljava.com>

FIM

Obrigado,
Maromo