



POO

Tratamento de exceção avançado

Prof. Alcides Calsavara


PUCPR




Conceitos



1. Comando try com mais de um catch
2. Comando try com multcatch
3. Comando try com catch polimórfico
4. Comando try com clausula finally
5. Hierarquia de classes para exceções em Java
6. Repasse de exceção entre métodos: throws
7. Emissão de uma exceção: throw
8. Exceções de aplicação




Comando **try** com
mais de um **catch**




```
int x = 0;  
int y = 10;  
int z = y/x;  
System.out.println(z);
```

```
String s = null;  
char c = s.charAt(2);  
System.out.println(c);
```


```
try {  
    int x = 0;  
    int y = 10;  
    int z = y/x;  
    System.out.println(z);  
  
    String s = null;  
    char c = s.charAt(2);  
    System.out.println(c);  
}  
catch (ArithmeticException e) {  
    System.out.println(  
        "Excecao em expressao aritmetica");  
}  
catch (NullPointerException e) {  
    System.out.println(  
        "Uso de referencia nula");  
}
```




Comando **try** com **multicatch**



```
try {  
    int x = 0;  
    int y = 10;  
    int z = y/x;  
    System.out.println(z);  
  
    String s = null;  
    char c = s.charAt(2);  
    System.out.println(c);  
}  
catch (ArithmeticException |  
        NullPointerException e) {  
    System.out.println("Deu algum problema");  
    e.printStackTrace();  
}
```

Comando **try** com **catch** polimórfico




```
try {  
    int x = 0;  
    int y = 10;  
    int z = y/x;  
    System.out.println(z);  
  
    String s = null;  
    char c = s.charAt(2);  
    System.out.println(c);  
}  
catch (Exception e) {  
    e.printStackTrace();  
}
```

java.lang.ArithmeticException: / by zero


java.lang.NullPointerException

Exception é a classe
mais genérica para
exceções.


Comando **try** com
mais de um **catch** e
com polimorfismo

```
try {  
    int x = 0;  
    int y = 10;  
    int z = y/x;  
    System.out.println(z);  
  
    String s = null;  
    char c = s.charAt(2);  
    System.out.println(c);  
}  
catch (ArithmeticException e) {  
    System.out.println(  
        "Excecao em expressao aritmetica");  
}  
catch (Exception e) {  
    e.printStackTrace();  
}
```

A ordem de catch é
da classe mais específica
para a mais genérica.




Comando **try** com
cláusula **finally**



```
class T {  
    public static void main(String[] args) {  
        try {  
            String s = null;  
            char c = s.charAt(3);  
            System.out.println(c);  
        }  
        catch (Exception e) {  
            e.printStackTrace();  
        }  
        finally {  
            System.out.println("Fim do programa");  
        }  
    }  
}
```

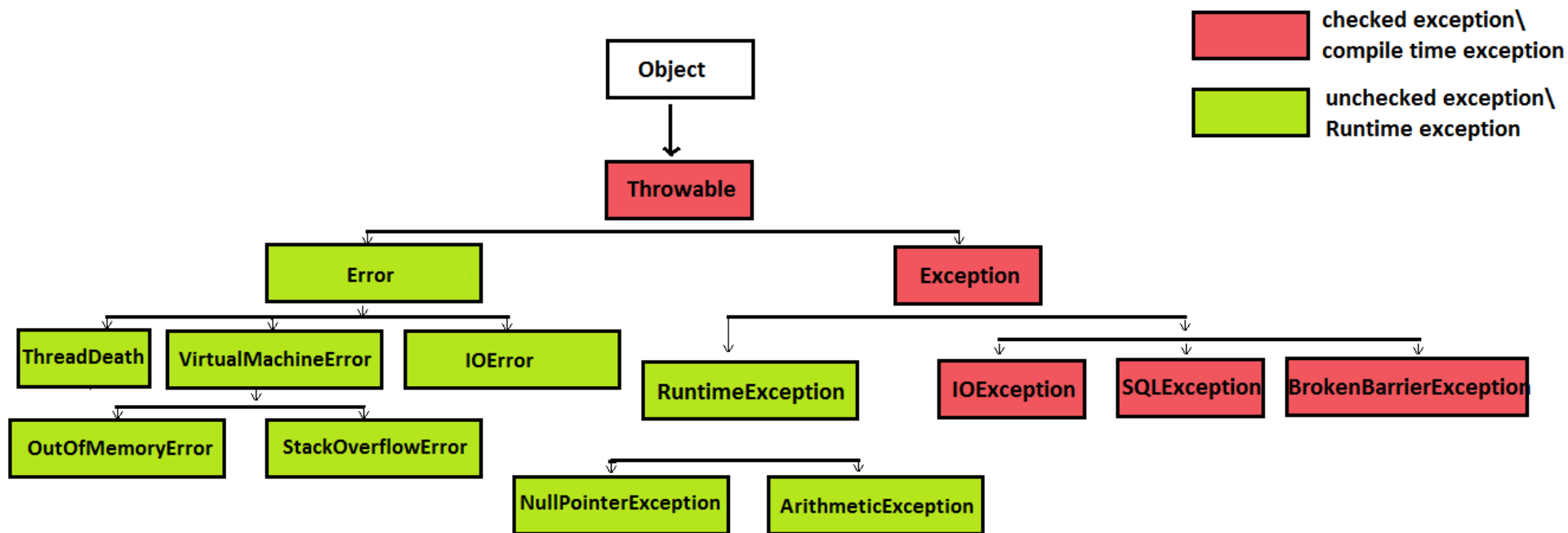
O bloco de comandos da cláusula finally é sempre executado, com ou sem a ocorrência de exceção.



Hierarquia de classes para exceções em Java

-- Exceções de Sistema --





unchecked exception : verificada somente em tempo de execução

checked exception : verificada em tempo de compilação e de execução


```
class T {  
    public static void main(String[] args) {  
        int x = ... // ler valor do teclado  
        int y = ... // ler valor do teclado  
        System.out.println( divide(x,y) );  
    }  
    private static int divide(int a, int b) {  
        int z = a/b;  
        return z;  
    }  
}
```

ArithmeticException é uma unchecked exception.

Embora haja uma potencial operação com divisão por zero, o compilador não exige que seja feito o tratamento de exceção (com uso de try-catch).

```
class T {  
    public static void main(String[] args) {  
        try {  
            FileReader arquivo =  
                new FileReader("dados.txt");  
            BufferedReader buffer =  
                new BufferedReader(arquivo);  
            String str;  
            while ((str = buffer.readLine()) != null) {  
                System.out.println(str);  
            }  
        }  
        catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

IOException é uma checked exception.

Toda operação sobre arquivos pode dar exceção.
Por isso, o compilador exige o tratamento de exceção.

Repasse da responsabilidade pelo tratamento de exceção

throws

throw

```
class T {  
    public static void main(String[] args) {  
        try {  
            ler();  
        }  
        catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
    private static void ler() throws IOException  
    {  
        FileReader arquivo =  
            new FileReader("dados.txt");  
        BufferedReader buffer =  
            new BufferedReader(arquivo);  
        String str;  
        while ((str = buffer.readLine()) != null) {  
            System.out.println(str);  
        }  
    }  
}
```

O método ler repassa a responsabilidade pelo tratamento de exceção para quem o chama.

```
class Main {  
    public static void main(String[] args) {  
        try { f(); }  
        catch (Exception e) { System.out.println("Excecao em main"); }  
        finally { System.out.println("finally em main"); }  
    }  
    private static void f() throws Exception  
    {  
        try  
        {  
            System.out.println("try");  
            throw new Exception();  
            System.out.println("fim do try");  
        }  
        catch(Exception e)  
        {  
            System.out.println("catch");  
        }  
        finally  
        {  
            System.out.println("finally");  
        }  
    }  
}
```

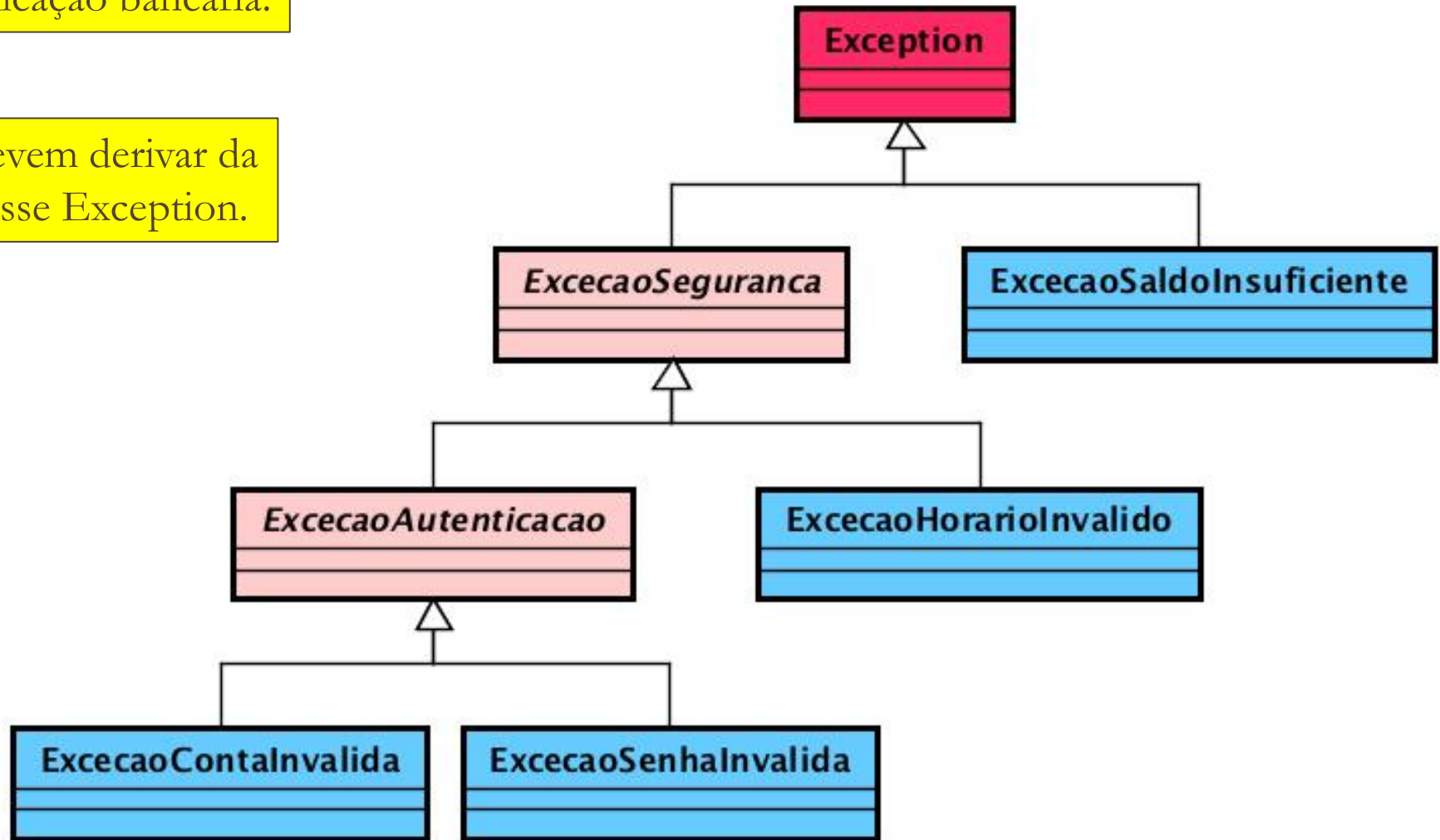


Exceções de Aplicação



Exceções em uma
aplicação bancária.

Devem derivar da
classe Exception.




```
class ExcecaoSaldoInsuficiente extends Exception {  
    public ExcecaoSaldoInsuficiente(String mensagem)  
{  
        super(mensagem);  
    }  
    public ExcecaoSaldoInsuficiente() {  
        super();  
    }  
}
```

Uma classe que representa uma exceção é uma classe como outra qualquer da aplicação. Logo, pode conter atributos e métodos conforme desejado.

Pode ser definido um construtor que recebe uma mensagem como parâmetro e repassa essa mensagem para a superclasse. A mensagem poderá ser lida chamando-se o método `getMessage()` para a instância da classe.

```
class ContaCorrente {  
    private double saldo;  
  
    public ContaCorrente(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public void retirar(double valor)  
        throws ExcecaoSaldoInsuficiente  
    {  
        if (saldo < valor)  
            throw new ExcecaoSaldoInsuficiente();  
        saldo = saldo - valor;  
    }  
}
```

O comando **throw** no método retirar lança uma exceção para o método que fez a chamada e interrompe a execução do método retirar.

```
class T {  
    public static void main(String[] args) {  
        ContaCorrente conta = new ContaCorrente(100);  
        try {  
            conta.retirar(200);  
        }  
        catch (ExcecaoSaldoInsuficiente e) {  
            e.printStackTrace();  
        }  
    }  
}
```

A chamada do método `retirar` para uma instância da classe `ContaCorrente` tem, obrigatoriamente, que ocorrer num bloco **try**.

```
class T {  
    public static void main(String[] args) {  
        ContaCorrente conta = new ContaCorrente(100);  
        try {  
            sacar(conta, 200);  
        }  
        catch (ExcecaoSaldoInsuficiente e) {  
            e.printStackTrace();  
        }  
    }  
    private static void sacar(ContaCorrente c, double v)  
        throws ExcecaoSaldoInsuficiente  
    {  
        c.retirar(v);  
    }  
}
```

Como o método `sacar` chama o método `retirar` para uma instância da classe `ContaCorrente` sem usar o comando **try**, a responsabilidade por tratar a exceção é repassada para o método que o chama, nesse caso, o método `main`.