POO – Módulo 8

Tratamento de exceção

Módulo ministrado pelo

Prof. Edson Emílio Scalabrin

PUCPR

Material produzido pelo
prof. Alcides Calsavara (BCC/PUCPR)

Conceitos

1. **Tratamento de exceção**
2. **Exceção como objeto**
3. **Hierarquia de classes de exceções**
    a. **Exceções de sistema**
    b. **Exceções de aplicação**
4. **Estrutura try-catch-finally**
5. **Repasse de exceção entre métodos: throws**
6. **Emissão de uma exceção: throw**

EXEMPLOS SIMPLES DE EXCEÇÕES

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

O que há de errado neste código?

Qual o resultado da sua execução?

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

O que há de errado neste código?

Divisão por zero.

Qual o resultado da sua execução?

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

O que há de errado neste código?

Divisão por zero.

Qual o resultado da sua execução?

Exception in thread "main" java.lang.ArithmeticException: / by zero

```
int[] valor = new int[5];
valor[1] = 120;
valor[2] = 180;
valor[3] = 200;
valor[4] = 250;
valor[5] = 300;
System.out.println("Fim");
```

O que há de errado neste código?

Qual o resultado da sua execução?

```
int[] valor = new int[5];
valor[1] = 120;
valor[2] = 180;
valor[3] = 200;
valor[4] = 250;
valor[5] = 300;
System.out.println("Fim");
```

O que há de errado neste código?

Indexação inválida.

Qual o resultado da sua execução?

```
int[] valor = new int[5];
valor[1] = 120;
valor[2] = 180;
valor[3] = 200;
valor[4] = 250;
valor[5] = 300;
System.out.println("Fim");
```

O que há de errado neste código?

Indexação inválida.

Qual o resultado da sua execução?

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5

```
String s = "Drone";
char c = s.charAt(5);
System.out.println(c);
```

O que há de errado neste código?

Qual o resultado da sua execução?

```
String s = "Drone";
char c = s.charAt(5);
System.out.println(c);
```

O que há de errado neste código?

Indexação inválida.

Qual o resultado da sua execução?

```
String s = "Drone";
char c = s.charAt(5);
System.out.println(c);
```

O que há de errado neste código?

Indexação inválida.

Qual o resultado da sua execução?

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 5

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

O que há de errado neste código?

Qual o resultado da sua execução?

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

O que há de errado neste código?

Uso de referência nula para chamada de método.

Qual o resultado da sua execução?

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

O que há de errado neste código?

Uso de referência nula para chamada de método.

Qual o resultado da sua execução?

Exception in thread "main" java.lang.NullPointerException

-- tal que o programa execute até o final --

TRATAMENTO DE EXCEÇÃO BASEADO EM TESTE

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

```
int x = 0;
int y = 10;
int z;
if (x!=0) {
    z = y / x;
    System.out.println(z);
}
else {
    System.out.println(“Divisao por zero");
}
```

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

```
s = null;
if (s!=null) {
    c = s.charAt(2);
    System.out.println(c);
}
else {
    System.out.println("Referência nula");
}
```

-- tal que o programa execute até o final --

TRATAMENTO DE EXCEÇÃO BASEADO EM TRY-CATCH

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

```
try {
    int x = 0;
    int y = 10;
    int z = y/x;
    System.out.println(z);
}
catch(ArithmeticException e) {
    System.out.println(
          "Excecao em expressao aritmetica");
}
```

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

```
try {
    String s = null;
    char c = s.charAt(2);
    System.out.println(c);
}
catch(NullPointerException e) {
    System.out.println(
                    "Uso de referencia nula");
}
```

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

```java
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 POLIMORFIMO

```
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.

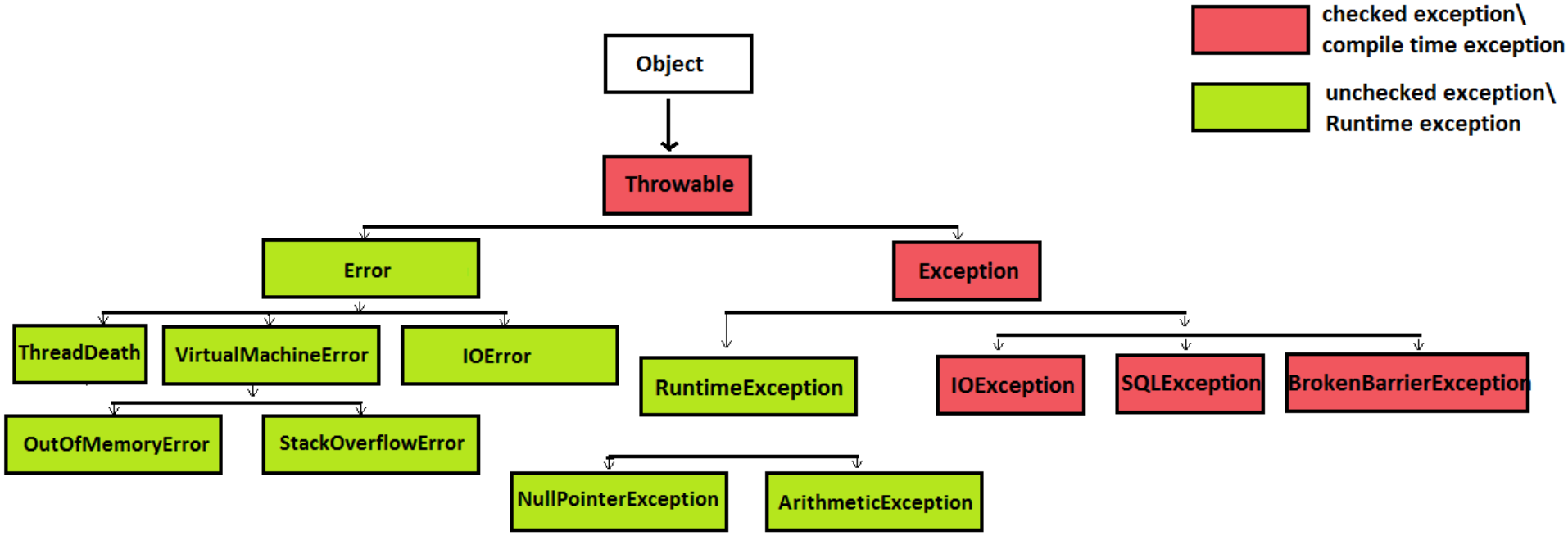
COMANDO TRY COM CLÁUSULA FINALLY

```
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.

-- Exceções de Sistema --

HIERARQUIA DE CLASSES PARA EXCEÇÕES EM JAVA



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

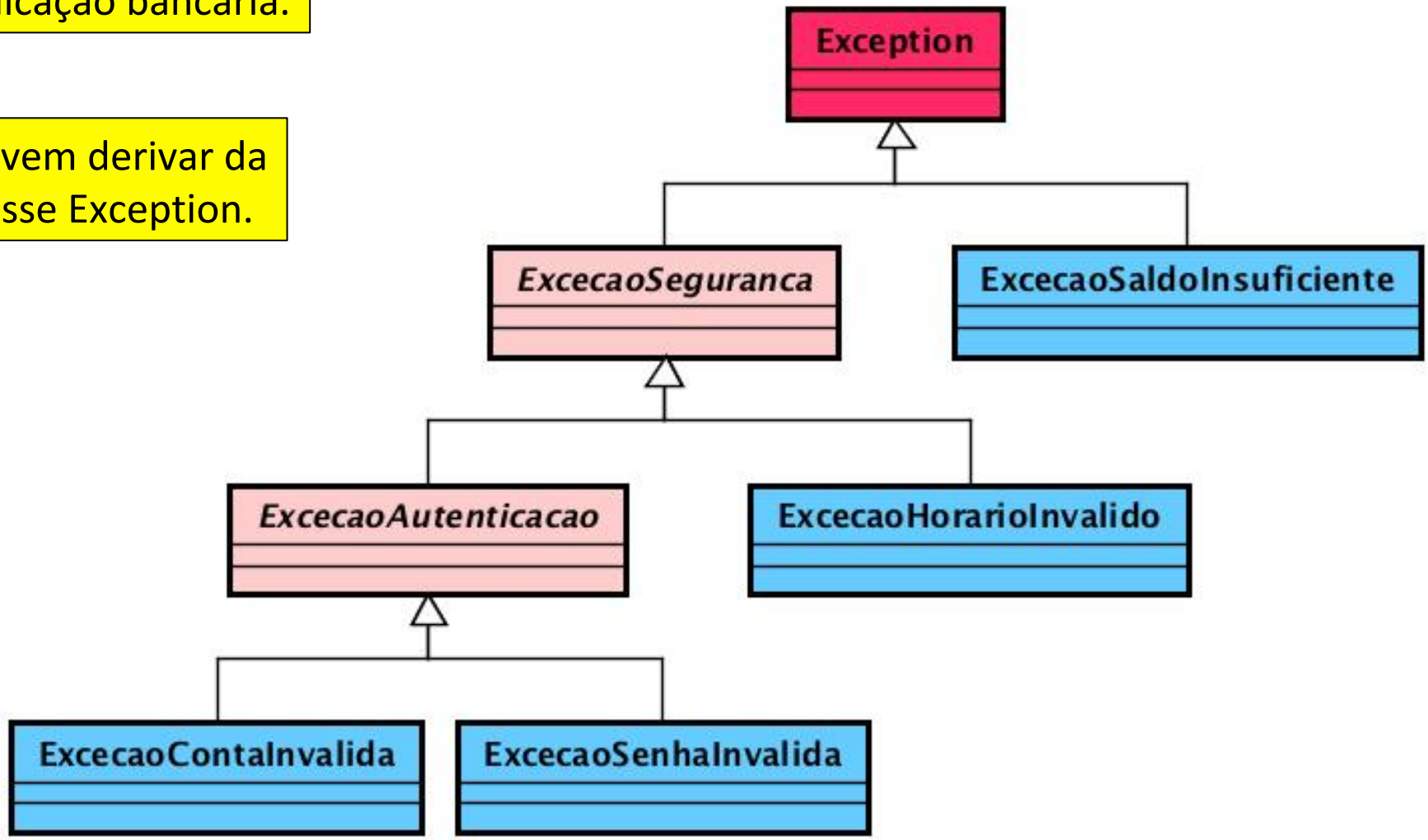
```
class T {
   public static void main(String[] args) {
      try {
         ler();
      }
      catch (IOException e) {
         e.printStackTrace();
      }
   }
   private static void ler() throws IOException
   {
      FileReader arquivo =
                     new FileReader("dados.txt");
      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 execeção para quem o chama.

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`.