

Programação Orientada a Objetos  
Fausto Maranhão Ayres

## 4

# Tratamento de exceção

## O que é Tratamento de Exceção

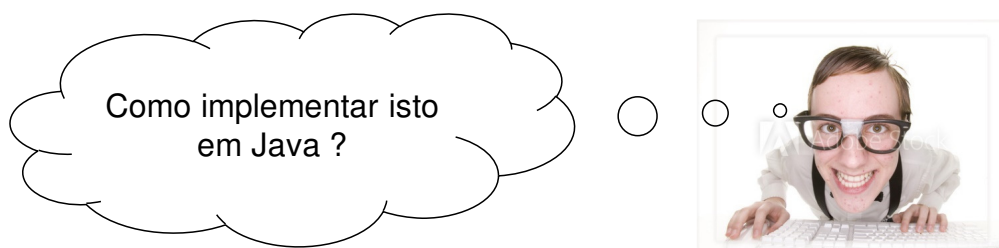
# O que é Tratamento de Exceção?

- O tratamento de exceção é o **procedimento de resposta** a situações indesejadas (exceções) que podem ocorrer durante a execução de um programa.
- Analogia: Cozinhar uma feijoada
  - Exceção:
    - Acabou o gás durante o cozimento
  - Tratamento da exceção:
    - Interromper o cozimento e comprar o gás.

3

## Exemplo 1

- Considere a classe **Conta** e o método **debitar()**:
  - Possível exceção (situação indesejável):  
“saldo ficar negativo”
  - Como prever esta situação no `debitar()` ?
    - Quando o valor a ser debitado é maior que o saldo



# Tratamento da exceção (meia boca)

- Interromper o debitar() e printar uma mensagem

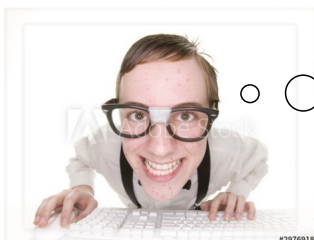
```
public class Conta{...  
    public void debitar(double quantia){  
        if(quantia > saldo) {  
            System.out.println("quantia incorreta");  
            return;  
        }  
        saldo = saldo - quantia;  
    }  
}
```

fausto.ayres@ifpb.edu.br

5

## Porque essa solução não é boa?

- Não é uma boa prática “printar” mensagens na **console** já que ela não aparecerá em outras plataformas (windows, web, mobile, etc)
  - O lugar ideal para exibir mensagens ao usuário é na Interface Gráfica, ou seja, na classe de **Aplicação**



Mas como exibir a msg na classe de Aplicação, se a exceção ocorre dentro do debitar() ?

fausto.ayres@ifpb.edu.br

6

# Ciclo do Tratamento de exceção (definitivo)

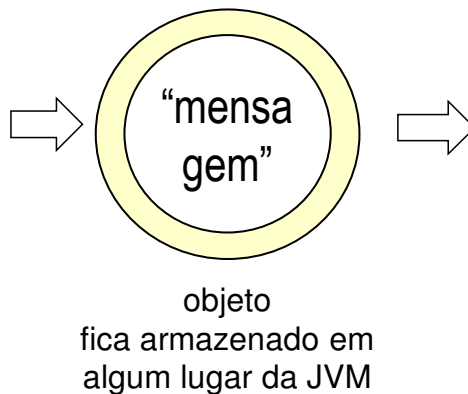
## Na classe **Conta**

```
void debitar(...) {
```

Se a exceção ocorrer  
Então

**interromper** o método  
e **lançar objeto** com a  
mensagem dentro

```
}
```



## Na classe **Aplicacao**

```
void main(...) {
```

**capturar objeto** e  
exibir a mensagem  
para o usuário  
(tratamento)

```
}
```

usuário  
  
desktop  
web  
app  
...

fausto.ayres@ifpb.edu.br

7

## Metáfora



8

# Sinalizar e Lançar exceção

## ■ No método debitar()

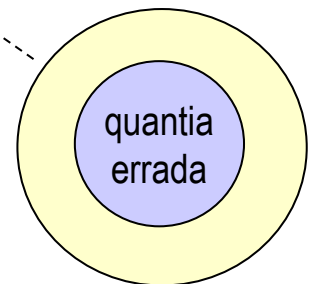


Sinalizar

```
public void debitar(double quantia) throws Exception
{
    if(quantia > saldo)
        throw new Exception("quantia errada");
    saldo = saldo - quantia;
}
```



Lançar



objeto  
Exception

O comando **throw** interrompe imediatamente a execução do método e **envia** o objeto para a JVM

fausto.ayres@ifpb.edu.br

9

# Detectar, Capturar e Tratar exceção

## ■ No método main():

```
public static void main(...) {
```

Detectar

```
try{
```

```
    Conta c1 = new Conta(...);
    c1.creditar(50);
    c1.debitar(70);
```

Capturar

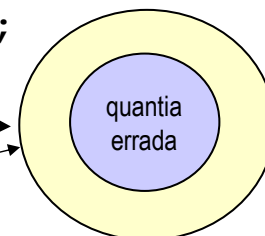
```
} catch(Exception e){
```

```
    System.out.println(e.getMessage());
```

```
}
```

Tratar

objeto  
Exception



throw  
catch

"quantia errada"

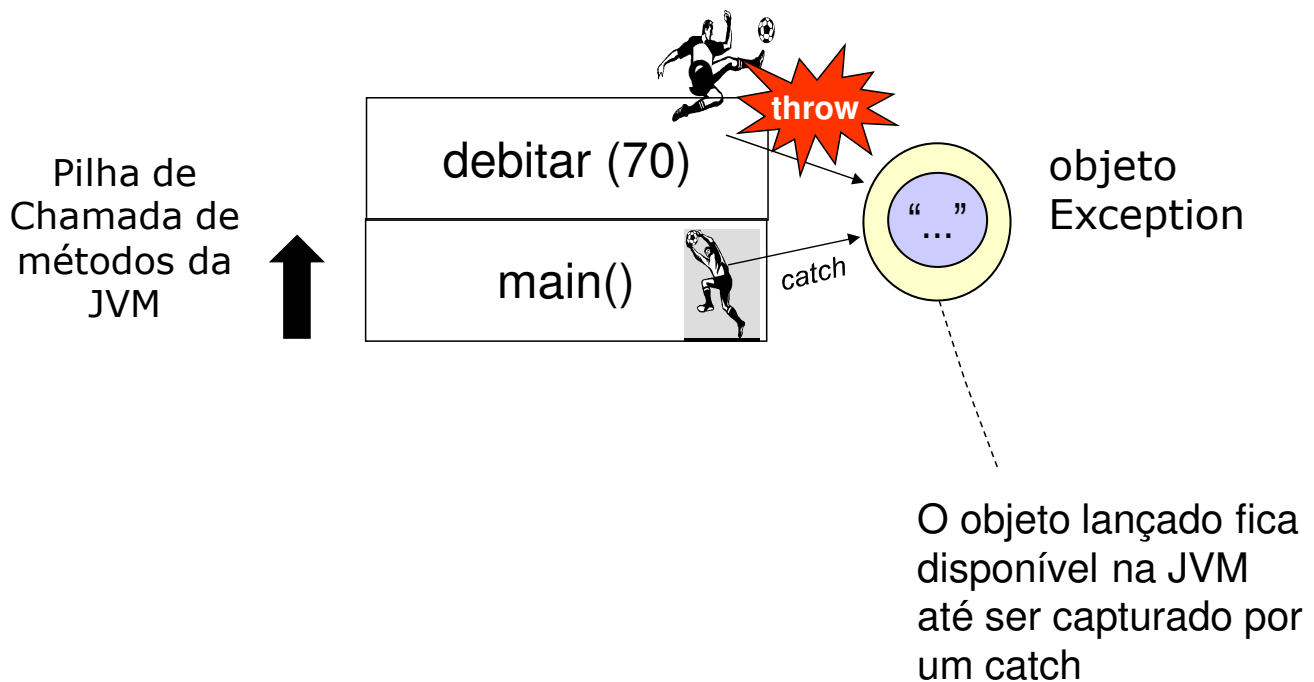


Caso ocorra o **throw**, o bloco **try** também é interrompido e o bloco **catch** captura o objeto através da variável **e**

fausto.ayres@ifpb.edu.br

10

# Interrupção do debitar()



fausto.ayres@ifpb.edu.br

11

## Interrupção do bloco try{}

- Após o **throw**, a execução do bloco **try** é interrompida e desviada para o bloco **catch**

```
public static void main(...) {  
    try{  
        Conta c1 = new Conta(111, ...);  
        c1.creditar(50);  
        c1.debitar(70);  
        c1.debitar(20); // não é executado  
    }  
    catch (Exception e) {  
        System.out.println(e.getMessage());  
    }  
}
```

interrupção

"quantia errada"

fausto.ayres@ifpb.edu.br

12

# try/catch individuais

- Boa prática é separar cada chamada do debitar()

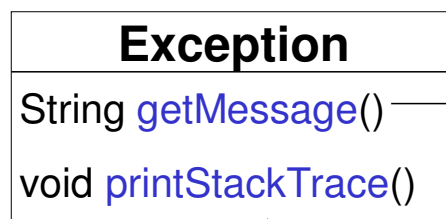
```
c1.creditar(50);  
try{  
    c1.debitar(70);    //não consegue debitar  
}  
catch(Exception e){  
    System.out.println(e.getMessage());  
}
```

```
try{  
    c1.debitar(20);    //debita normalmente  
}  
catch(Exception e){  
    System.out.println(e.getMessage());  
}
```

fausto.ayres@ifpb.edu.br

13

## A classe Exception



acessa a  
mensagem interna

printa na console o histórico da pilha de chamadas  
até o momento do lançamento (throw)

É importante na fase de depuração

fausto.ayres@ifpb.edu.br

14

# Exemplo de `printStackTrace()`

```
public static void main(...) {  
    try{  
        Conta c1 = new Conta(111,...);  
        c1.creditar(50);  
        c1.debitar(70);           //interrupção  
    }  
    catch (Exception e ){  
        e.printStackTrace();  
    }  
}
```

Exception in thread "main" java.lang.Exception: quantia incorreta  
at Conta.debitar(Conta.java:25)  
at Teste.main(Teste.java:15)

Nome da classe      mensagem

interrupção

## Desvio de exceção



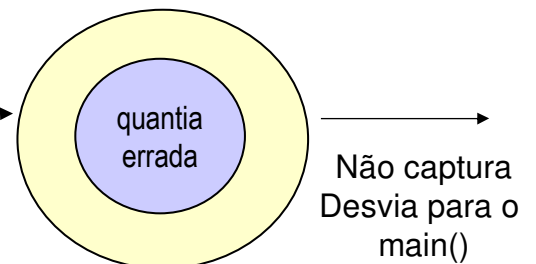
# Desviar a exceção

- Se um método não capturar (catch) uma exceção lançada por outro método, ele o repassa para o próximo catch disponível, através do **throws**
- Exemplo: `main()` → `transferir()` → `debitar()`
  - O método `transferir()` não captura a exceção recebida do `debitar()`, mas a desvia para o método `main()`

## Exemplo: método `transferir()`

```
public void transferir(double quantia, Conta destino)  
    throws Exception
```

```
    this.debitar(quantia);  
    destino.creditar(quantia);  
}
```



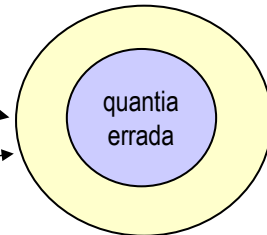
# Detectar, capturar e tratar

- O **main** trata a exceção desviada pelo **transferir()**

```
public static void main(...) {  
    try{  
        Conta c1 = new Conta (111,...);  
        Conta c2 = new Conta (222,...);  
        c1.creditar(50);  
        c1.transferir(100,c2);  
    }
```

```
    catch (Exception e){  
        System.out.println(e.getMessage());  
    }
```

```
}
```



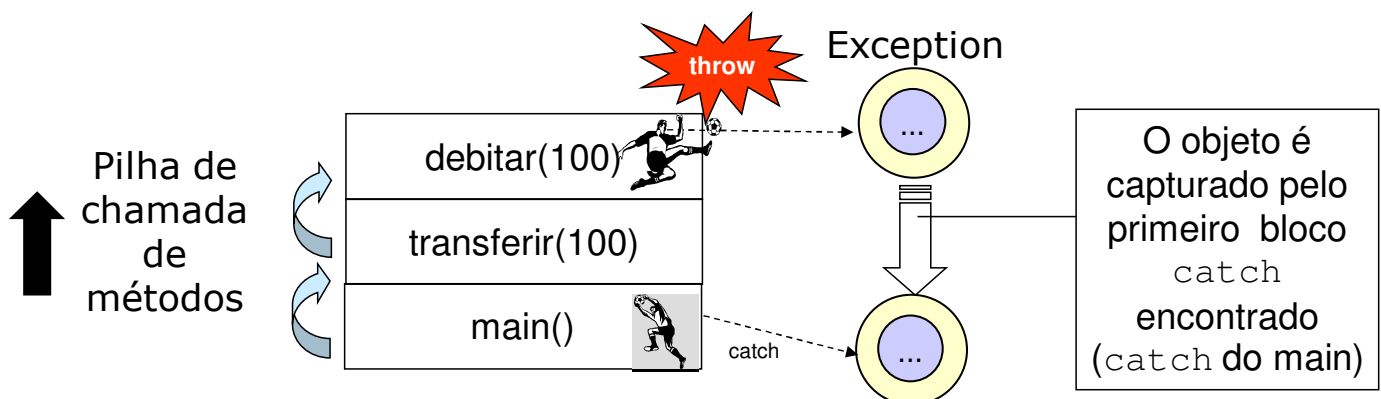
"quantia errada"

fausto.ayres@ifpb.edu.br

19

## Desvio no transferir()

- A exceção lançada no **debitar()** é capturada pelo **main()**

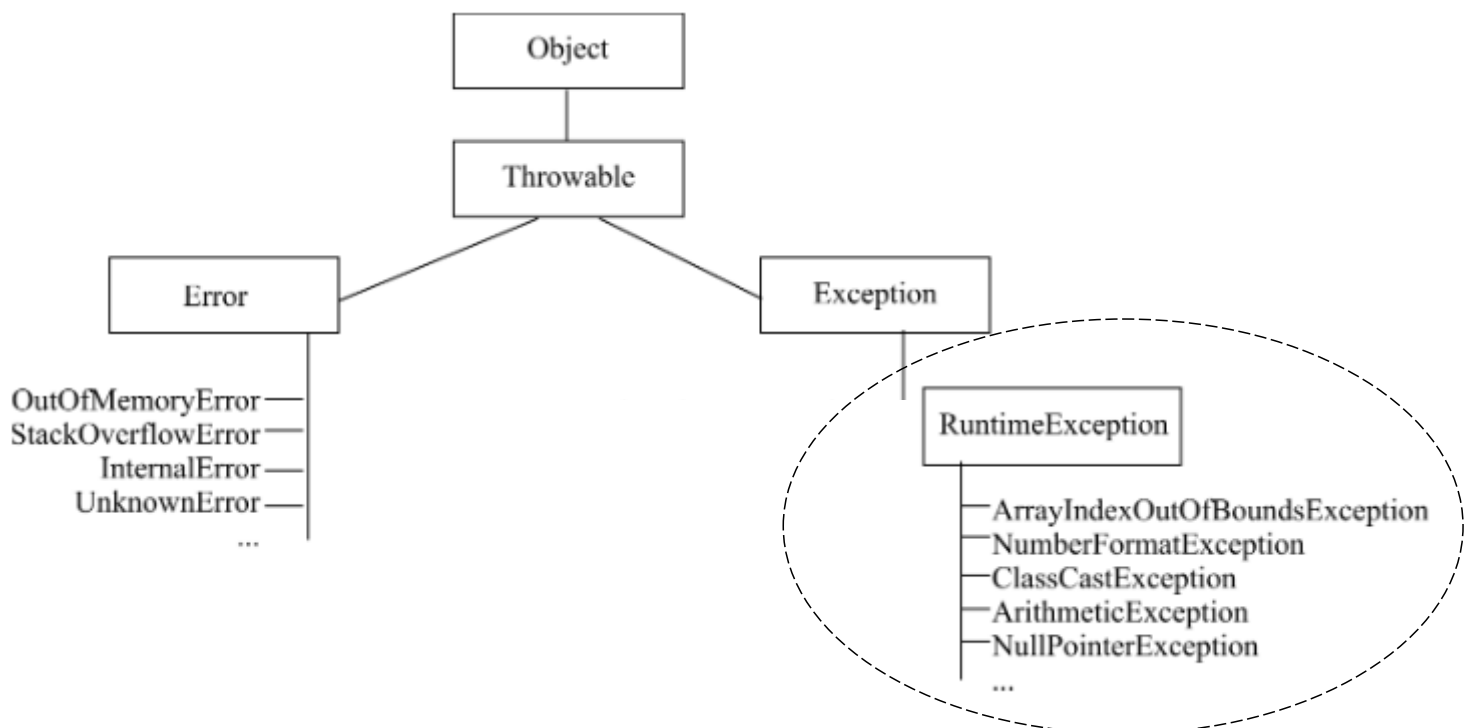


fausto.ayres@ifpb.edu.br

20

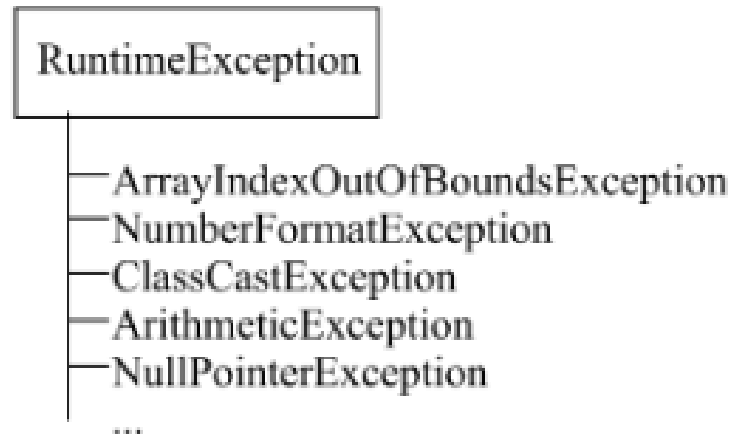
# Exceções *Runtime* que causam interrupção do programa

## Principais classes de exceção do Java



# O que são classes de exceção Runtime?

- São classes de exceção lançadas por vários métodos da API Java em reação a **erros graves (bug)** de programação



- A captura deste tipo de exceção é **opcional**, mas caso não seja feita, o programa será “**abortado**”

## Classe `ArrayIndexOutOfBoundsException`

- Esta exceção é lançada quando há uso incorreto de índices

```
String[] nomes = {"bala", "bola", "balão"};
System.out.println(nomes[5]); //aborta
```

# Classe NullPointerException

- Esta exceção é lançada na falta de objeto na chamada de método

```
Retangulo r = null;  
double x = r.calcularArea(); //aborta
```

# Classe NullPointerException

O programa web abaixo foi abortado devido ao lançamento de uma NullPointerException - causada por um bug dentro do programa

Home / Minha NET / Fatura / Fatura Digital

Error opening /minha\_net/minha\_assinatura/cobranca/fatura\_email/fatura\_email.jsp.

The source of this error is:

java.lang.NullPointerException

```
at java.lang.String.startsWith(String.java:1421)  
at java.lang.String.startsWith(String.java:1450)  
at br.com.netnaveb.web.taglib.cms.CMSContentParameter.doEndTag(CMSContentParameter.java:100)  
at jsp_servlet._minha_net._minha_assinatura._cobranca._fatura_email._fatura_email._jsp_tag24(_fatura_email.java:1438)  
at jsp_servlet._minha_net._minha_assinatura._cobranca._fatura_email._fatura_email._jspService(_fatura_email.java:383)  
at weblogic.servlet.jsp.JspBase.service(JspBase.java:34)  
at weblogic.servlet.internal.StubSecurityHelper$ServletServiceAction.run(StubSecurityHelper.java:227)  
at weblogic.servlet.internal.StubSecurityHelper.invokeServlet(StubSecurityHelper.java:125)  
at weblogic.servlet.internal.ServletStubImpl.execute(ServletStubImpl.java:292)  
at weblogic.servlet.internal.TailFilter.doFilter(TailFilter.java:26)
```

< VOLTAR PARA MINHA NET

# Classe NumberFormatException

- Lançada pelo método **parseInt()** devido a erro de conversão string para int

```
public static int parseInt(String s)
    throws NumberFormatException
{...
}
```

Exemplos:

```
int i = Integer.parseInt("23"); //ok
int j = Integer.parseInt("2t"); //aborta
```

# Classe NumberFormatException

O programa abaixo não captura a exceção:

```
2 public class Teste {
3     public static void main(String[] args) {
4         Scanner teclado = new Scanner(System.in);
5         System.out.println("digite um numero:");
6         String texto = teclado.nextLine();
7         int numero = Integer.parseInt(texto);
8     }
9 }
```

digite um numero:

x

O programa aborta

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "x"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatEx
    at java.base/java.lang.Integer.parseInt(Integer.java:660)
    at java.base/java.lang.Integer.parseInt(Integer.java:778)
    at Teste.main(Teste.java:7)
```

# Captura de NumberFormatException

O programa abaixo captura a exceção:

```
Scanner teclado = new Scanner(System.in);
try {
    System.out.println("digite um numero:");
    String texto = teclado.nextLine();
    int numero = Integer.parseInt(texto);
}
captura catch (NumberFormatException e1) {
    System.out.println("numero invalido");
}
```

```
digite um numero:
x
numero invalido
```

*O programa não aborta !!*

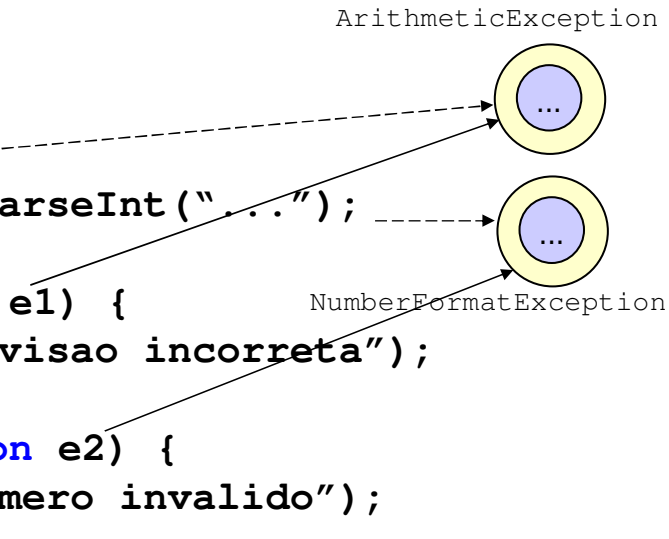
fausto.ayres@ifpb.edu.br

29

## Capturando diferentes exceções

- Um método pode capturar várias exceções com vários catch

```
try
{
    ...
    double media = 100/n;
    int numero = Integer.parseInt("...");
}
catch (ArithmeticException e1) {
    System.out.println("divisao incorreta");
}
catch (NumberFormatException e2) {
    System.out.println("numero invalido");
}
```



ArithmeticException

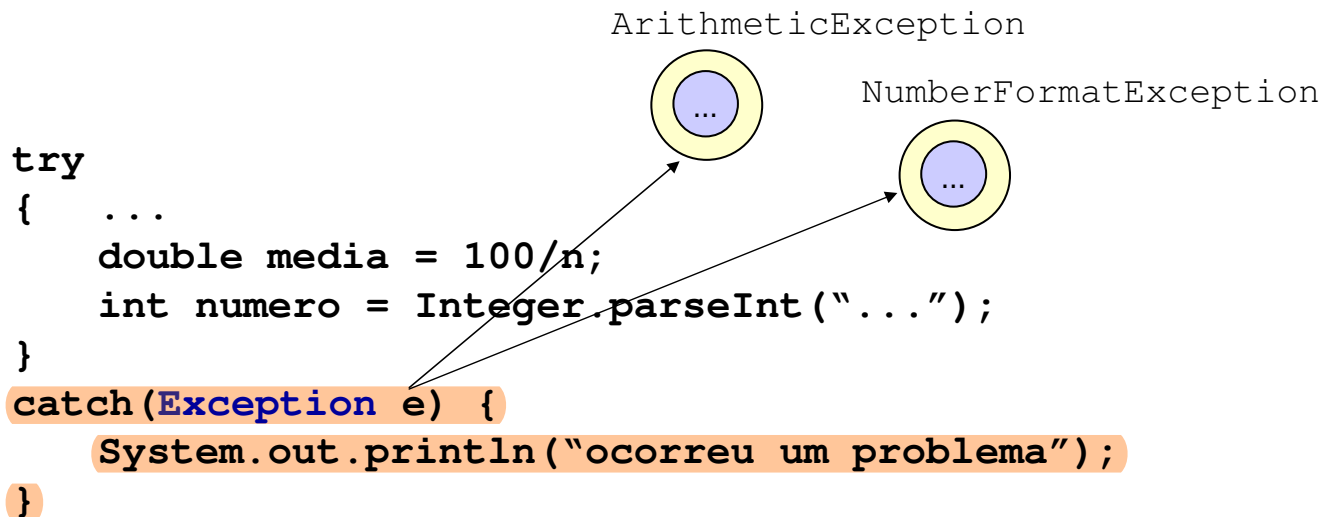
NumberFormatException

Cada **catch** captura somente um tipo de exceção.

fausto.ayres@ifpb.edu.br

30

# Capturando exceção genérica



A classe **Exception** pode ser usada no lugar de qualquer outra classe. Como resultado disso, o tratamento passa a ser mais **genérico**.

## Lançando vários tipos de exceção

- Um método pode lançar várias exceções, desde que as respectivas classes de exceção sejam declaradas no throws

```
public void metodo() throws Exception1, Exception2{
    if (condicao1)
        throw new Exception1();
    if (condicao2)
        throw new Exception2();
    ...
}
```



## Obs:

- O bloco **finally** é opcional
- É executado em seguida ao try ou catch, mesmo que haja um return;

```
try{  
    ...  
    return;  
}  
catch (Exception e) {  
    ...  
    return;  
}  
finally {  
    ...  
}
```