

# PROGRAMAÇÃO ORIENTADA A OBJETOS

## Exceções em Java



# Exceção (Exception)

2

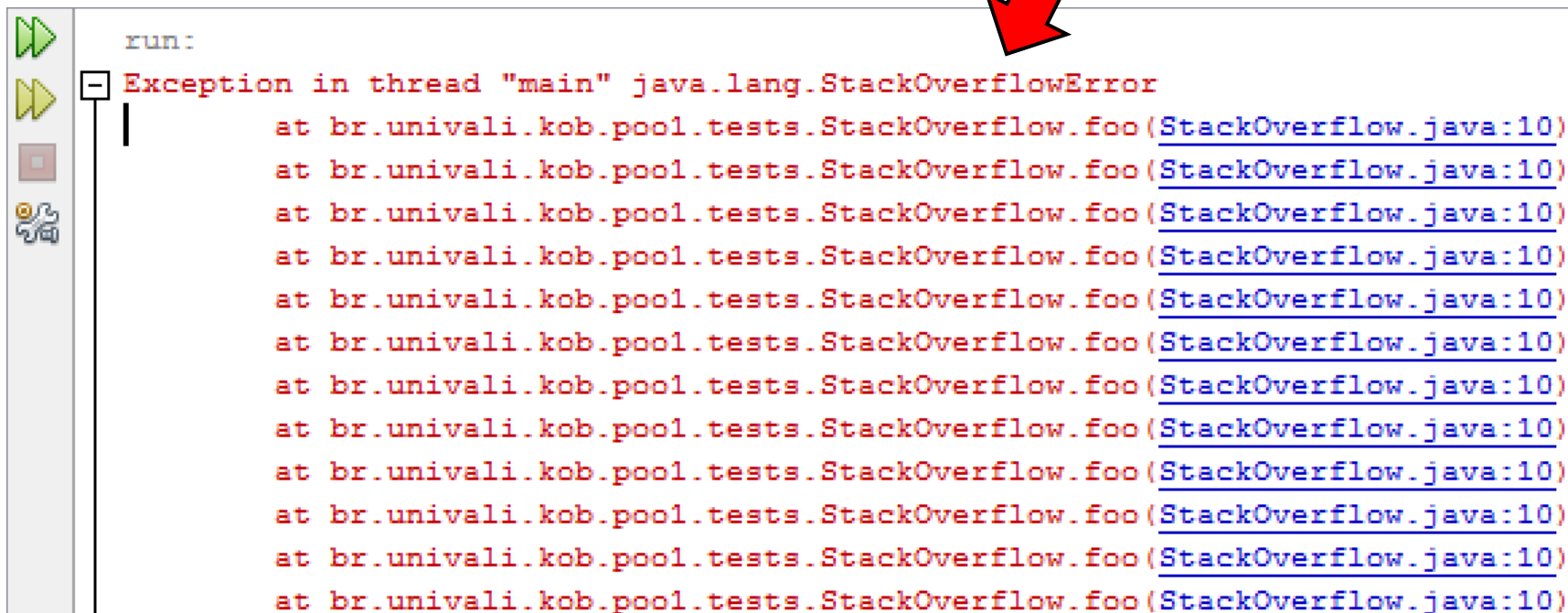
- Evento que ocorre durante a **execução de um programa** e que **interrompe o fluxo normal** de instruções

```
public class StackOverflow {  
    public void foo() {  
        foo();  
    }  
}
```

The method foo may recurse if not overridden in subclasses  
----  
(Alt-Enter shows hints)

# Chamada recursiva sem ponto de parada

```
new StackOverflow().foo();
```



# Exemplos de Exceção

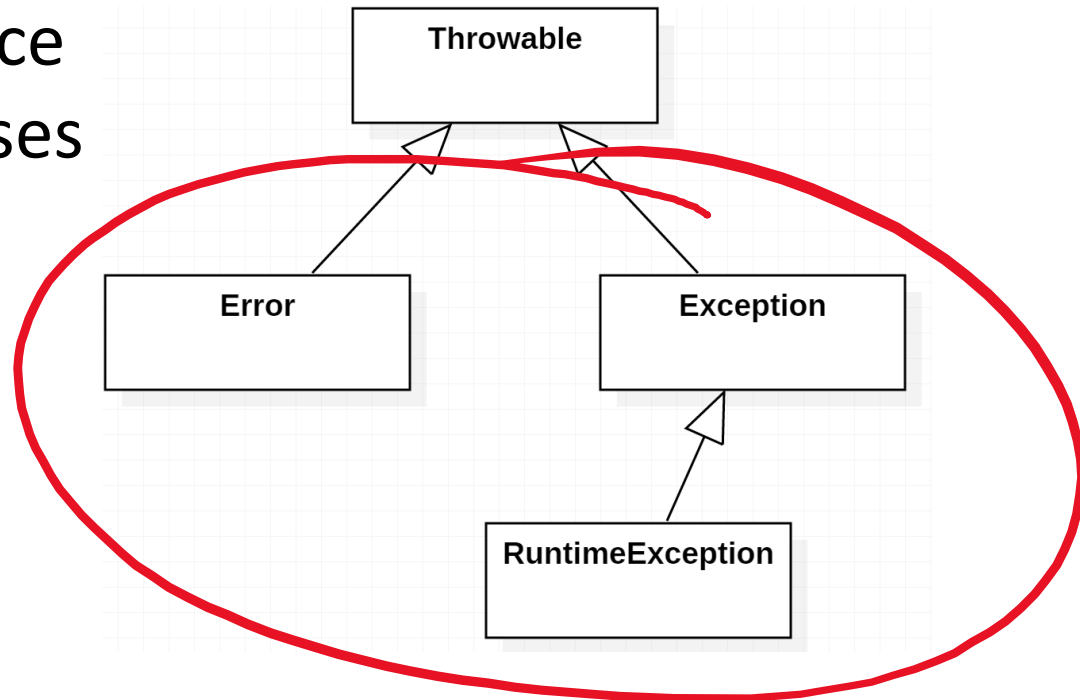
4

- ❑ Divisão por zero
- ❑ Operação matemática inválida
- ❑ Tentativa de acesso a uma posição inválida em um vetor
- ❑ Tentativa de acesso a um objeto que não foi criado
- ❑ etc.

# Exceções em Java

5

A linguagem Java oferece uma hierarquia de classes de exceção



# Documentação sobre as classes

6

## ❑ Classe Throwable

❑ <http://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html>

## ❑ Classe Exception

❑ <http://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html>


## ❑ Classe RuntimeException

❑ <http://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html>

## ❑ Classe Error

❑ <http://docs.oracle.com/javase/8/docs/api/java/lang/Error.html>

# Exceções em Java

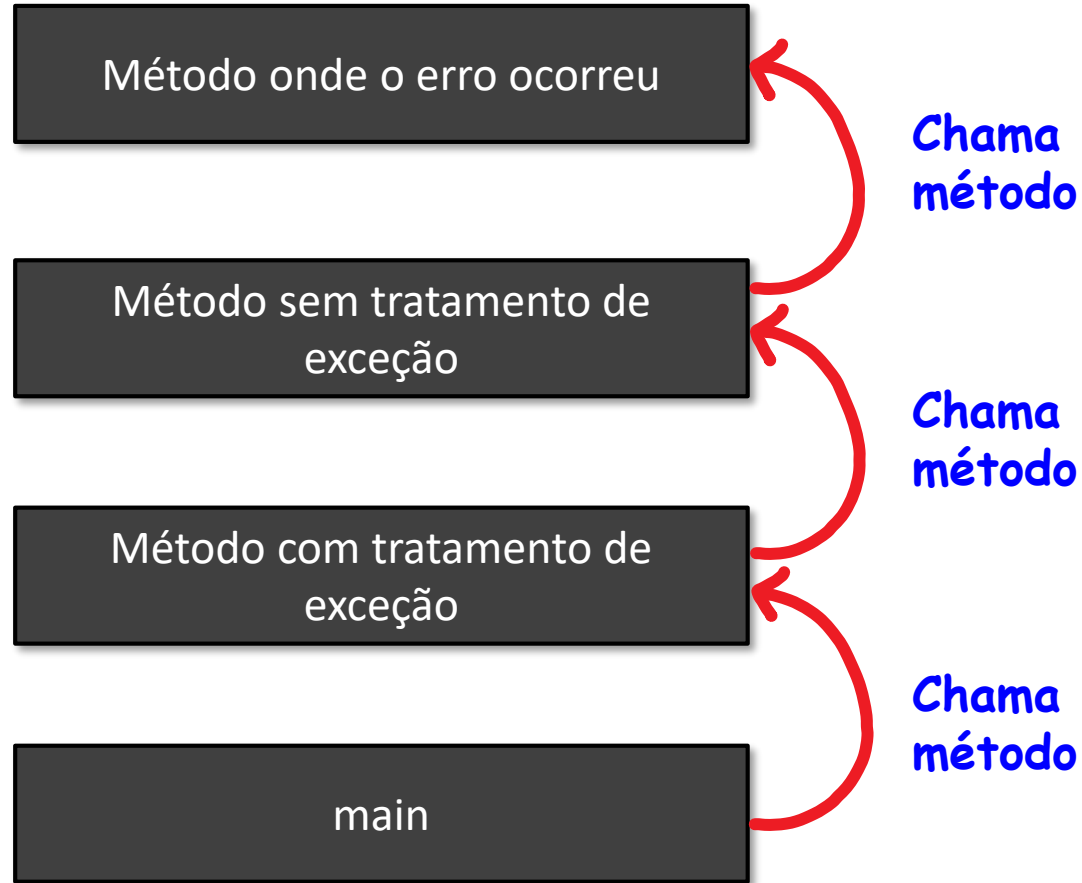
- ▶▶▶  7
- ❑ Quando um erro ocorre dentro de um método, o próprio método cria um **objeto de exceção** e o entrega para o ambiente de execução
  - ▣ Disparar (“**throwing**”) uma exceção
  - ▣ O objeto criado contém informações sobre o erro, incluindo seu tipo e o estado do programa quando o erro ocorreu

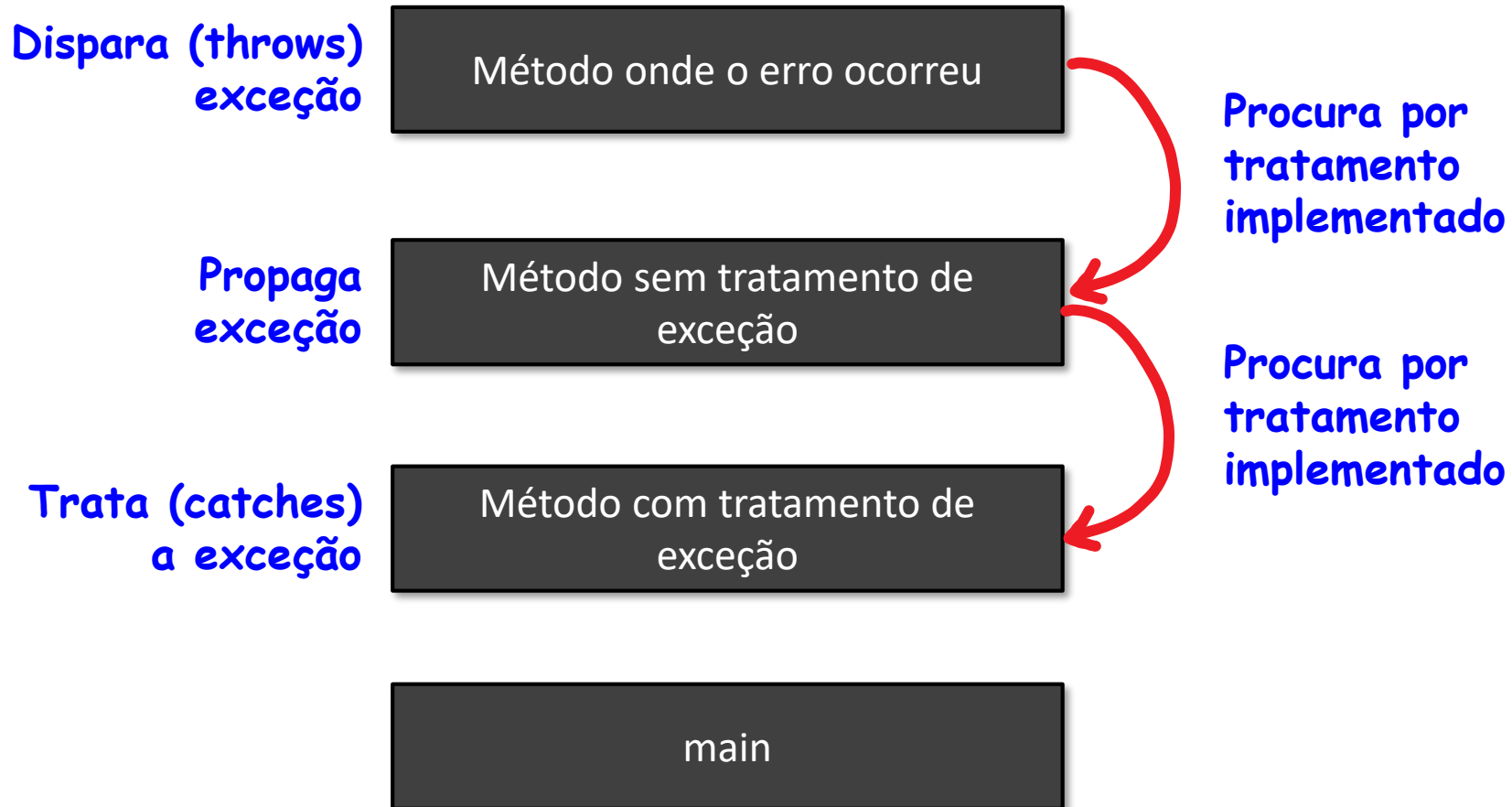
# Exceções em Java



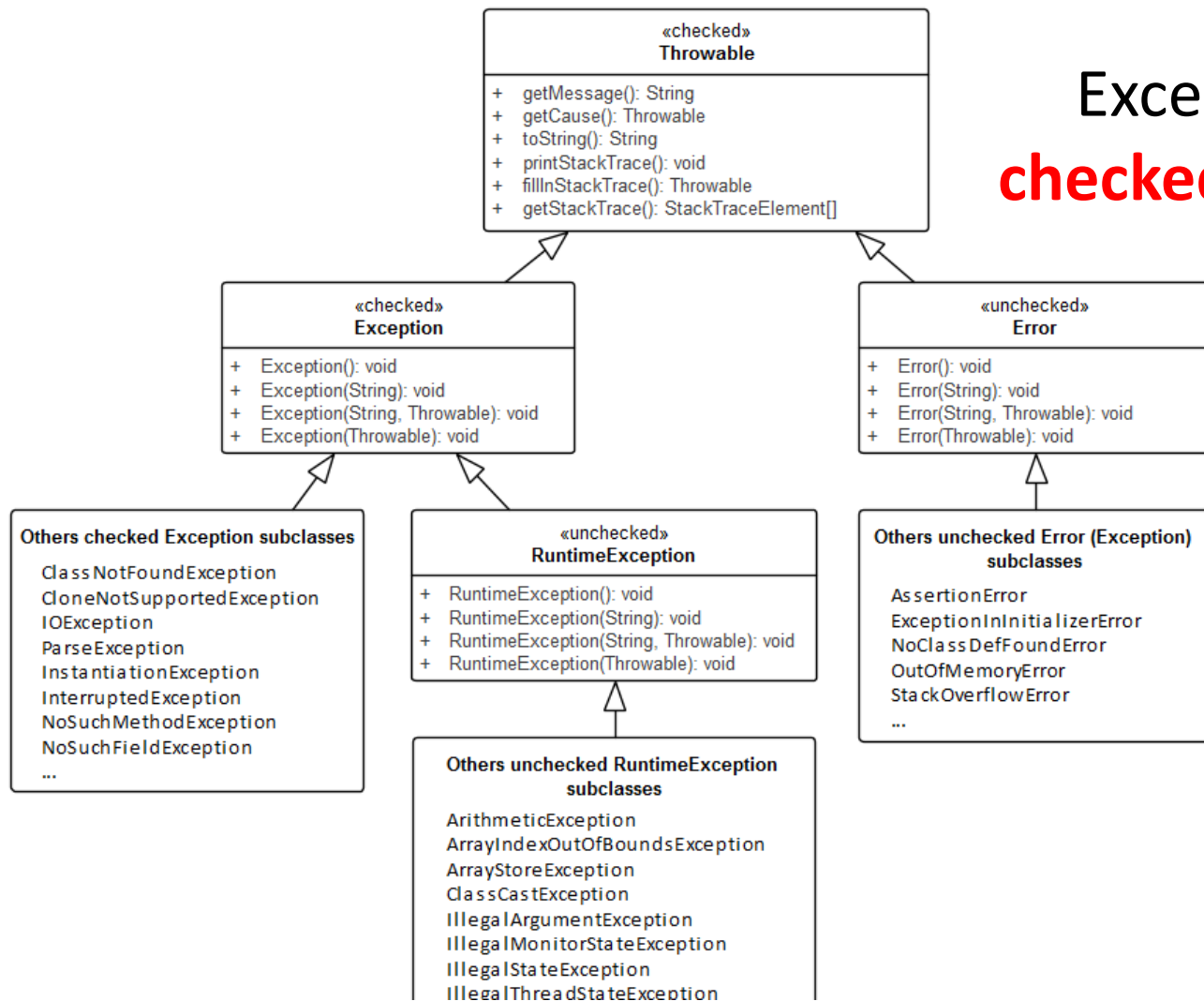
- Depois que um método dispara (*throws*) uma exceção, o ambiente de execução (JVM) tenta encontrar algum tratamento para tratá-la
- ▣ A sequência de possíveis tratamentos para manipular a exceção é a pilha dos métodos que foram chamados para chegar até o método onde o erro ocorreu







# Exceções podem ser **checked** ou **unchecked**



# Tipos de exceção

12

## ❑ Checked

- ▣ Checadas em tempo de compilação
- ▣ É necessário tratá-las (capturando ou propagando)

## ❑ Unchecked

- ▣ Não são checadas em tempo de compilação, apenas em tempo de execução
- ▣ Não é necessário fornecer um tratamento

# Exception (checked)

13

- ❑ Condições excepcionais que uma aplicação bem escrita deveria antecipar
  - ▣ A aplicação deve implementar ações para se recuperar destas condições (ex: usuário solicita a abertura de um arquivo inexistente)
  - ▣ Portanto, o tratamento de exceções é obrigatório para este tipo de exceção

# RuntimeException (unchecked)

14

- ❑ Condições excepcionais que são "internas" a uma aplicação que não podem ser antecipadas
  - ▣ Tipicamente, a aplicação não tem como se recuperar destas condições
    - Exemplo: defeitos de programação
  - ▣ Evitar tratamento, permitindo que o defeito ocorra
    - **Permite encontrar e eliminar defeitos**

# Error (unchecked)

15

- ❑ Condições excepcionais que são "externas" a uma aplicação que não podem ser antecipadas
  - ▣ Tipicamente, a aplicação não tem como se recuperar destas condições
    - Exemplo: falha de hardware
  - ▣ Evitar tratamento
    - Se for defeito, é possível identificá-lo e corrigi-lo
      - Ex: StackOverflowError

# Tratamento de exceções

16

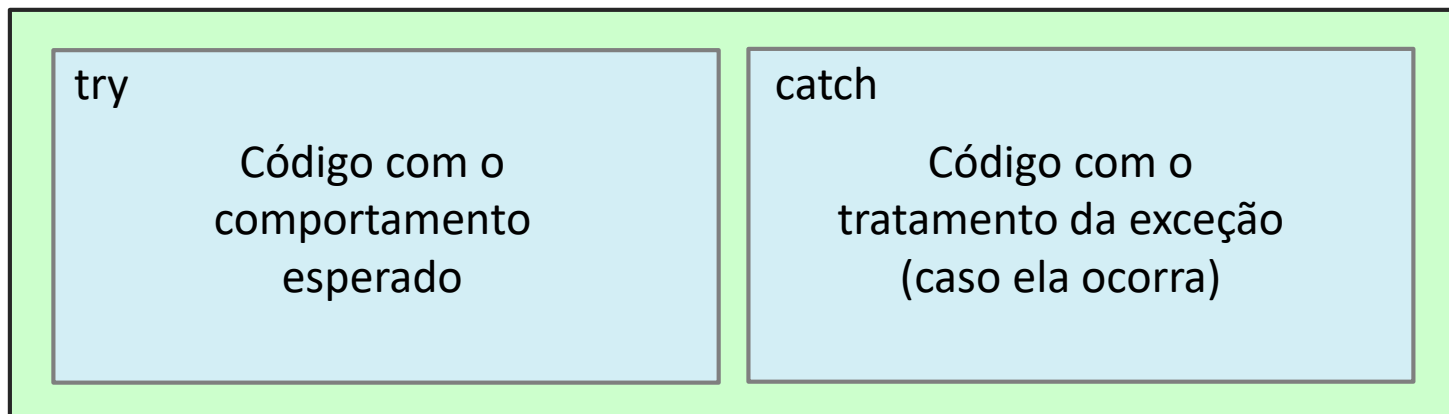
- ❑ Permite que a execução do programa continue mesmo após a ocorrência de uma exceção
  - ▣ Pode aumentar a robustez do programa
  - ▣ Mas a dica é postergar ao máximo
- ❑ Adote o princípio “**dispare mais cedo, capture mais tarde**”



# Tratamento de exceções

17

- Separa o código que define comportamento "normal" (esperado) do código que realiza tratamento de exceções



# try...catch

18

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```

1. O código controlado pela cláusula **try** é executado
2. Se ocorrer uma exceção, o controle é desviado para o código dentro da cláusula **catch**
3. Se a exceção ocorrida estiver sendo considerada pela cláusula **catch**, o código de tratamento da exceção é executado (caso contrário, a exceção é repassada ao ambiente de execução)
4. Se não ocorrer uma exceção, o tratamento da exceção (**catch**) não é executado

# try...catch

19

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```


- ❑ Você pode adicionar quantas cláusulas **catch** forem necessárias
  - ▣ Permite tratamento específico para cada tipo de exceção considerado
- ❑ **objExcecao** é o objeto de exceção criado quando a exceção ocorre
  - ▣ Ele possui várias informações sobre a exceção

# try...catch

20

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
}  
catch (TipoExcecao2 | TipoExcecao3 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```

Você pode considerar  
mais de uma exceção  
em uma mesma  
cláusula **catch**



# try...finally

21

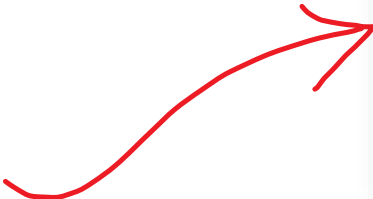
```
try {  
    <bloco de código>  
} finally {  
    <código sempre executado>  
}
```

1. O código controlado pela cláusula **try** é executado
2. Se ocorrer uma exceção, o controle é desviado para o código dentro da cláusula **finally** que é executado
3. Se não ocorrer uma exceção, após o último comando dentro da cláusula **try**, o código dentro da cláusula **finally** é executado (ou seja, o código dentro da cláusula **finally** é sempre executado)

# Combinando **try...catch...finally**

22

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
} finally {  
    <código sempre executado>  
}
```



Este bloco é  
executado  
mesmo quando uma  
exceção ocorrer!!

# Experimente...

23

```
public class OrdinalNumber {  
  
    private final String[] ORDINAL_SET = {"First", "Second", "Third"};  
  
    public String getFromCardinal(int cardinalNumber) {  
        try {  
            // Decrementa antes, pois o primeiro índice do vetor é ZERO  
            return ORDINAL_SET[--cardinalNumber];  
        } catch (ArrayIndexOutOfBoundsException exception) {  
            // A mensagem presente no objeto de exceção é o próprio índice  
            // usado. Note que o valor é o número cardinal menos UM, pois  
            // havíamos decrementado antes de acessar o vetor.  
            return "Invalid number (" + exception.getMessage() + ")";  
        }  
    }  
  
    public static void main(String[] args) {  
        OrdinalNumber ordinal = new OrdinalNumber();  
        System.out.println(ordinal.getFromCardinal(1)); // ok  
        System.out.println(ordinal.getFromCardinal(4)); // exceção  
    }  
}
```



```
run:  
First  
Invalid number (3)  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# A cláusula **throws**


24

- ❑ Permite especificar exceções que podem ser disparadas ou propagadas dentro de um determinado método
  - ▣ Você pode forçar (disparar) uma exceção dentro de um método, usando a cláusula **throw**
  - ▣ Você pode não desejar tratar a exceção no método atual e apenas propagá-la para o próximo método
    - O próximo método pode tratar a exceção ou continuar propagando





# Disparando uma exceção manualmente

25



```
public String getFromCardinal(int cardinalNumber) throws ArrayIndexOutOfBoundsException {  
    // Dispara a exceção antes da JVM  
    if (cardinalNumber < 1 || cardinalNumber > ORDINAL_SET.length) {  
        throw new ArrayIndexOutOfBoundsException(cardinalNumber);  
    }  
    // Decrementa antes, pois o primeiro índice do vetor é ZERO  
    return ORDINAL_SET[--cardinalNumber];  
}
```



```
run:  
First  
- Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 4  
  |   at br.univali.kob.pool.tests.OrdinalNumber.getFromCardinal(OrdinalNumber.java:22)  
  |   at br.univali.kob.pool.tests.OrdinalNumber.main(OrdinalNumber.java:31)  
  |   C:\Users\marce\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

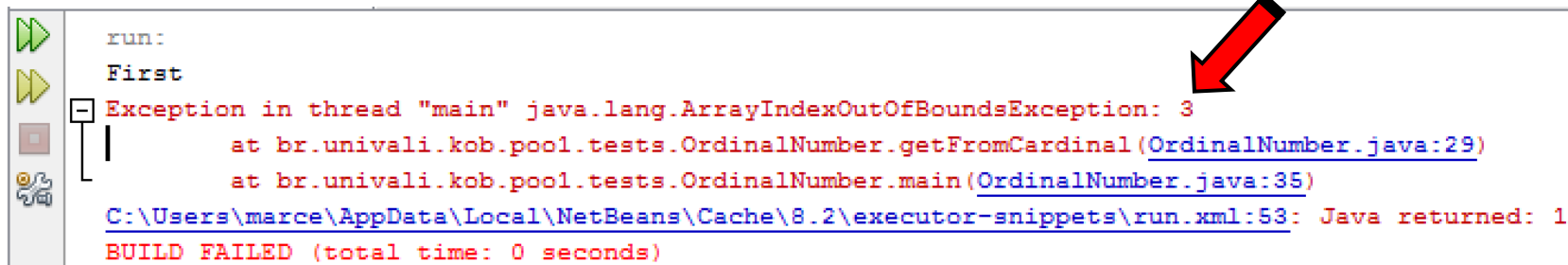
# Mas poderia ser mais simples...

26

**ArrayIndexOutOfBoundsException é unchecked (RuntimeException)**

```
public String getFromCardinal (int cardinalNumber) {  
    return ORDINAL_SET[--cardinalNumber];  
}
```

Mas observe que a mensagem pode ficar confusa (o objeto chamador passou 4 como argumento)



# try-with-resources

27

- Cláusula **try** (Java 7+) que declara um ou mais recursos
  - ▣ Um recurso é um objeto que deve ser fechado depois que o programa não precisar mais dele
    - Garante que cada recurso é fechado ao final da cláusula

```
public static void exemploTryWithResources() throws IOException {  
    try (BufferedReader br =  
        new BufferedReader(new FileReader("d:\\a.txt"))) {  
        System.out.println(br.readLine());  
    }  
}
```

# Criando sua própria classe de exceção

28

- ❑ Você pode ainda criar suas próprias exceções
  - ❑ Verifique se a plataforma Java já não oferece uma exceção que atenda às suas necessidades
    - ... ou alguma exceção já criada por outros programadores e fornecedores
  - ❑ Verifique se a nova exceção será realmente útil
  - ❑ Utilize a palavra Exception ao final do nome da classe criada (boas práticas de codificação Java)

```
public class OutOfRangeException extends IllegalArgumentException {  
  
    /** Valor avaliado que está fora da faixa esperada ...3 lines */  
    private final int value;  
    /** Texto que descreve o valor ...3 lines */  
    private final String valueLabel;  
    /** Valor mínimo definido pela faixa ...3 lines */  
    private final int min;  
    /** Valor máximo definido pela faixa ...3 lines */  
    private final int max;  
  
    /**  
     * A mensagem é montada com o rótulo passado no parâmetro.  
     *  
     * @param value o valor avaliado que está fora da faixa esperada  
     * @param valueLabel o texto que descreve o valor  
     * @param min o valor mínimo aceitável para a faixa  
     * @param max o valor máximo aceitável para a faixa  
     */  
    public OutOfRangeException(int value, String valueLabel, int min, int max) {  
        super("Value " + value + " for " + valueLabel + " is out of range [" +  
            min + ".." + max + "]);  
        this.value = value;  
        this.valueLabel = valueLabel;  
        this.min = min;  
        this.max = max;  
    }  
  
    /**  
     * Retorna o valor avaliado que está fora da faixa esperada  
     * +
```

# PROGRAMAÇÃO ORIENTADA A OBJETOS

## Exceções em Java

