

# Fundamentos de Java

## Tratamento de Exceções



Softblue  
cursos online

---

---

---


---

---

---

---

Tópicos Abordados



- Avisando sobre falhas em métodos
  - Booleanos
  - Magic numbers
- Exceções
  - Checked exceptions
  - Unchecked exceptions
- Lançando exceções
- Tratando exceções
- Transformando exceções
- Assertions

---

---

---


---

---

---

---

Avisando Sobre Falhas em Métodos



- Você precisa avisar quem chamou o método e informar que o método não executou como deveria. Como fazer?
  - As abordagens mais comuns são
    - Usar booleanos
    - Usar magic numbers

---

---

---

---

---

---

---

## Problema de Usar Booleanos



```
boolean sucesso = o.processar();  
  
if(sucesso) {  
    //código em caso de sucesso  
} else {  
    //código em caso de falha  
}
```

E se o retorno  
não for tratado?

O que  
falhou?

---

---

---

---

---

---

---

## Problema de Usar Magic Numbers



```
int resultado = o.processar();  
  
if(resultado == 100) {  
    //sucesso  
} else if (resultado == 110) {  
    //falha na validação  
} else if (resultado == 120) {  
    //falha de gravação no arquivo  
}
```

E se o retorno  
não for tratado?

Como entender este código sem  
uma tabela de códigos de erro?

---

---

---

---

---

---

---

## Exceções



- Exceções representam algo estranho ao sistema que normalmente não ocorre
- Em Java, o tratamento de exceções é feito por um código diferente do código executado quando não ocorre a exceção

---

---

---

---

---

---

---

## Classes Que Representam Exceções



- Exceções são representadas por classes
- As classes devem herdar direta ou indiretamente de *Exception*
- O Java tem classes que representam diversos tipos de exceção, mas o programador pode criar exceções específicas de acordo com a necessidade

---

---

---

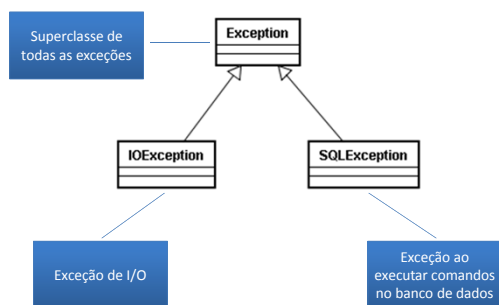
---

---

---

---

## Exemplos de Exceções



---

---

---

---

---

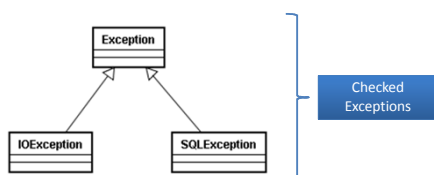
---

---

## Checked Exceptions



- Herdam direta ou indiretamente de *Exception*
- Só não podem herdar de *RuntimeException*



---

---

---


---

---

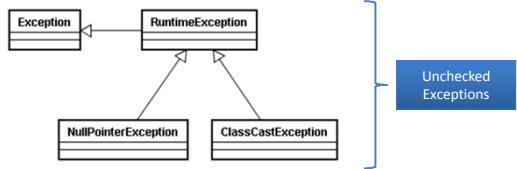
---

---

Unchecked Exceptions



- Também chamadas de *runtime exceptions*
- Herdam direta ou indiretamente de *RuntimeException*



---

---

---


---

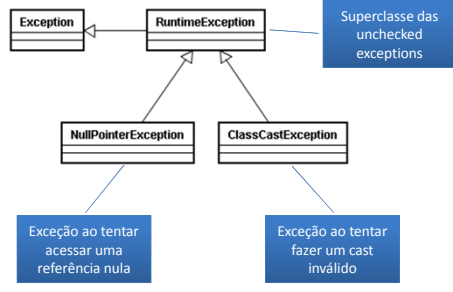
---

---

---

Unchecked Exceptions





---

---

---


---

---

---

---

Lançando Exceções



- O lançamento de exceções é feito através do *throw*

```
public void fazerAlgo() throws Exception {  
    throw new Exception();  
}
```

O *throw* é usado para lançar a exceção

O *throws* indica que o método pode lançar a exceção

---

---

---


---

---

---

---

Lançando Exceções



- É possível também lançar subclasses da exceção declarada pelo *throws*

```
public void fazerAlgo() throws Exception {  
    throw new IOException();  
}
```

IOException é uma  
subclasse de Exception

A declaração *throws Exception*  
está de acordo com a exceção  
lançada pelo método

---

---

---

---


---

---

---

---

Tratando Exceções



- Exceções podem ser tratadas através do uso do bloco *try/catch*
  - Determinado código tenta (*try*) executar um método e, caso alguma exceção aconteça, ele pega (*catch*) a exceção ocorrida e faz o que deseja
- Após uma exceção ter alcançado o bloco *catch*, o código volta o seu fluxo normal de execução

---

---

---

---


---

---

---

---

Tratando Exceções



```
public void m1() throws Exception {  
    throw new Exception();  
}
```

```
public void m2() {  
    try {  
        m1();  
    } catch (Exception e) {  
        ...  
    }  
    ...  
}
```

Se uma *Exception* acontecer,  
o fluxo é desviado para o  
bloco *catch*

Ao fim do bloco *catch*, a  
execução continua após o bloco

---

---

---

---

---


---

---

---

5

## Tratando Múltiplas Exceções



```
public void m1() throws IOException, SQLException {
    ...
}
```

```
public void m2() {
    try {
        m1();
    } catch (IOException e) {
        ...
    } catch (SQLException e) {
        ...
    }
    ...
}
```

Dependendo da exceção, o bloco *catch* correspondente é executado

No máximo um bloco *catch* é executado

---

---

---

---

---

---

---

---

## Multi-Catch



- É possível fazer o catch de mais de uma exceção ao mesmo tempo

```
try {
    m();
} catch (MyException1 e) {
    ...
} catch (MyException2 e) {
    ...
} catch (MyException3 e) {
    ...
}
```

```
try {
    m();
} catch (MyException1 | MyException2 | MyException3 e) {
    ...
}
```

---

---

---

---


---

---

---

---

## Ordem das Exceções no Catch



```
public void m1() throws IOException {
    throw new IOException();
}
```

```
public void m2() {
    try {
        m1();
    } catch (Exception e) {
        ...
    } catch (IOException e) {
        ...
    }
    ...
}
```

Toda exceção será tratada por este bloco *catch*

Este bloco *catch* nunca será executado

---

---

---

---

---

---

---

---

## Tratando e Lançando Exceções



```
public void m1() throws IOException, SQLException {  
    ...  
}
```

```
public void m2() throws IOException {  
    try {  
        m1();  
    } catch (SQLException e) {  
        ...  
    }  
    ...  
}
```

Apenas a `SQLException` é tratada.  
A `IOException` é lançada para  
quem chamou `m2()`

---

---

---

---

---

---

---

## Transformando Exceções



```
public void m1() throws IOException {  
    ...  
}
```

```
public void m2() throws AppException {  
    try {  
        m1();  
    } catch (IOException e) {  
        throw new AppException();  
    }  
    ...  
}
```

A `IOException` é relançada  
como uma `AppException`

---

---

---

---

---

---

---

## Lançando Unchecked Exceptions



- Estas exceções normalmente são provocadas por problemas de programação, não devendo ser tratadas
- Por este motivo, o Java não obriga o programador a tratar as unchecked exceptions

---

---

---

---

---

---

---

### Lançando Unchecked Exceptions

```
public void m1() {  
    throw new NullPointerException();  
}
```

O `throws` não é necessário, pois `NullPointerException` é unchecked

```
public void m2() {  
    m1();  
}
```

Não é necessário tratar a exceção. Caso ela ocorra, será lançada automaticamente para quem chamou `m2()`

---

---

---

---

---

---

---

### Algumas Considerações

- Se uma exceção for lançada pelo método `main()`, a JVM termina
- Exceções muito genéricas dificultam no entendimento do problema

---

---

---

---

---

---

---

### Assertions

- Garantir qualidade do código, executando testes que permitem validar a lógica e as suposições sobre o programa
- São usadas em tempo de desenvolvimento e desabilitadas em produção

---

---

---

---

---

---

---



## Exemplo



- Exemplo de utilização

```
public void metodo(int arg) {  
    assert arg > 0;  
    ...  
}
```

- Se a assertion falhar, a JVM lançará um *AssertionError*

---

---

---

---

---

---

---

## Mais um Exemplo



- Outro exemplo

```
public void metodo(int arg) {  
    assert arg > 0 : "arg menor que 0";  
    //...  
}
```

- Funciona como o exemplo anterior, mas a string fornecida será passada no construtor do *AssertionError*

---

---

---

---

---

---

---

## Habilitando Assertions



- Por padrão, as *assertions* ficam desabilitadas
- Para habilitá-las, é preciso passar o parâmetro **-ea** ao iniciar a JVM

```
java -ea MinhaClasse
```

---

---

---

---

---

---

---

## Considerações sobre Assertions



- Não use assertions para validar parâmetros de métodos públicos
  - Métodos públicos têm um comportamento bem definido sobre o que ocorre na passagem de parâmetros
  - Este comportamento deve ser o mesmo, havendo assertions ou não
- Não escreva assertions que interfiram na execução da aplicação
  - Estando a assertion habilitada ou não, a aplicação deve funcionar da mesma forma

---

---

---

---

---

---

---



---

---

---

---

---

---

---