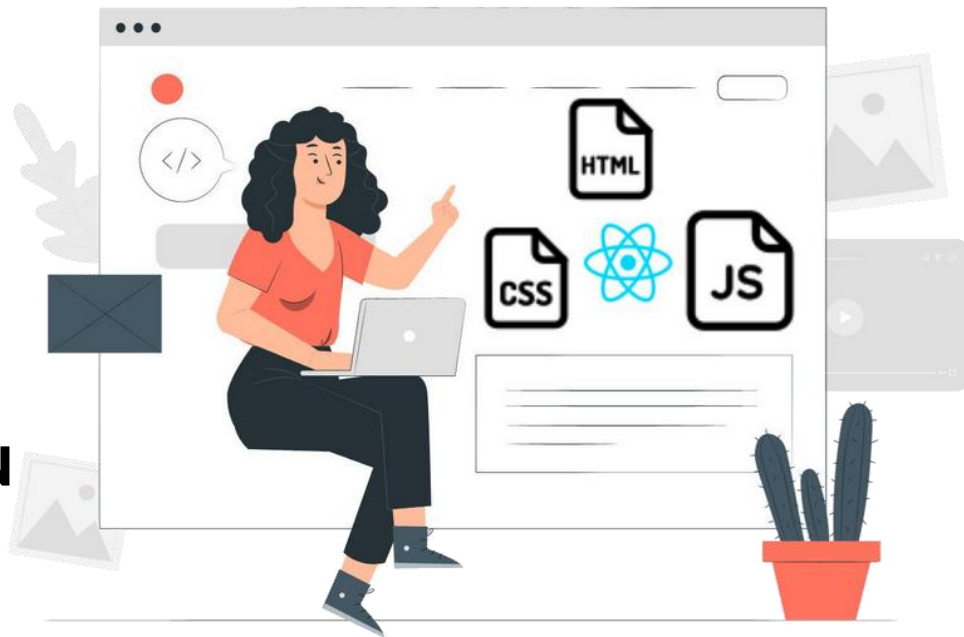


DOMAIN DRIVEN DESIGN

Eliane Marion



FERRAMENTAS DE TRABALHO



JAVA

Linguagem de programação orientada a objetos e amplamente utilizada



INTELLIJ

Ambiente de desenvolvimento integrado (IDE) usado na programação de computadores

MÉTODOS AVALIATIVOS



CHECKPOINTS

Exercícios ou projetos práticos desenvolvidos em laboratório.



Global Solution

Avaliação prática abordando assuntos trabalhados em aula.



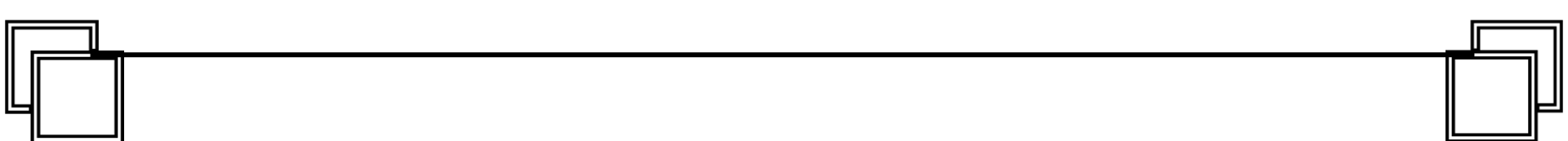
Challenge

Entrega bimestral do projeto seguindo orientações.

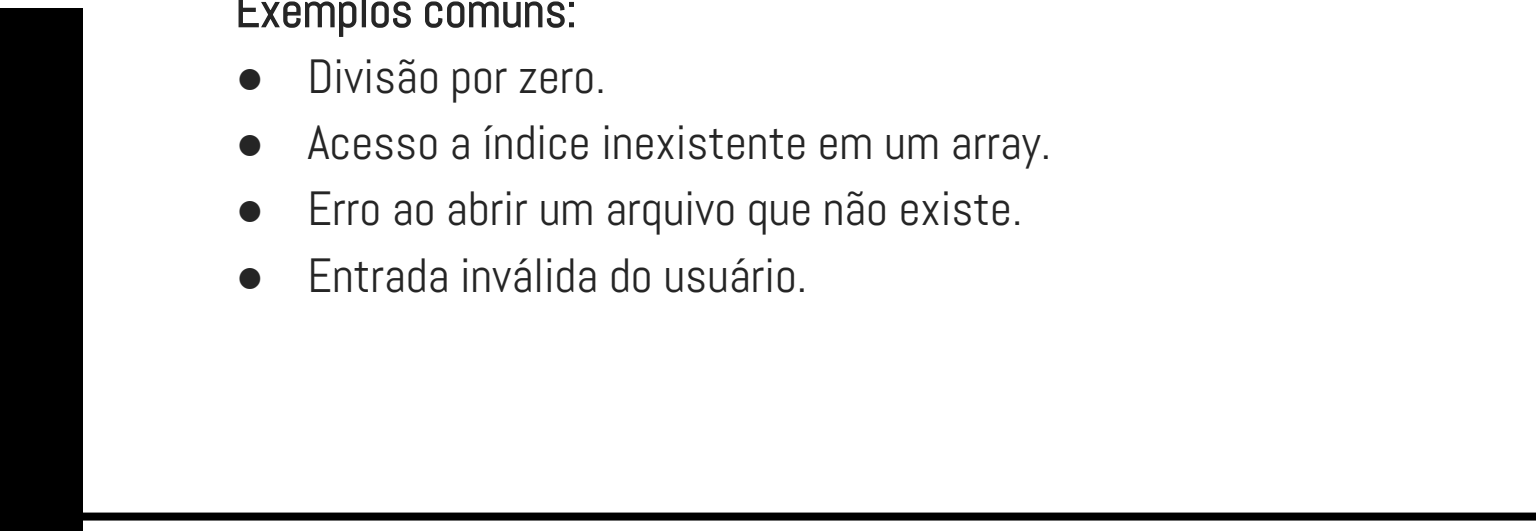


01

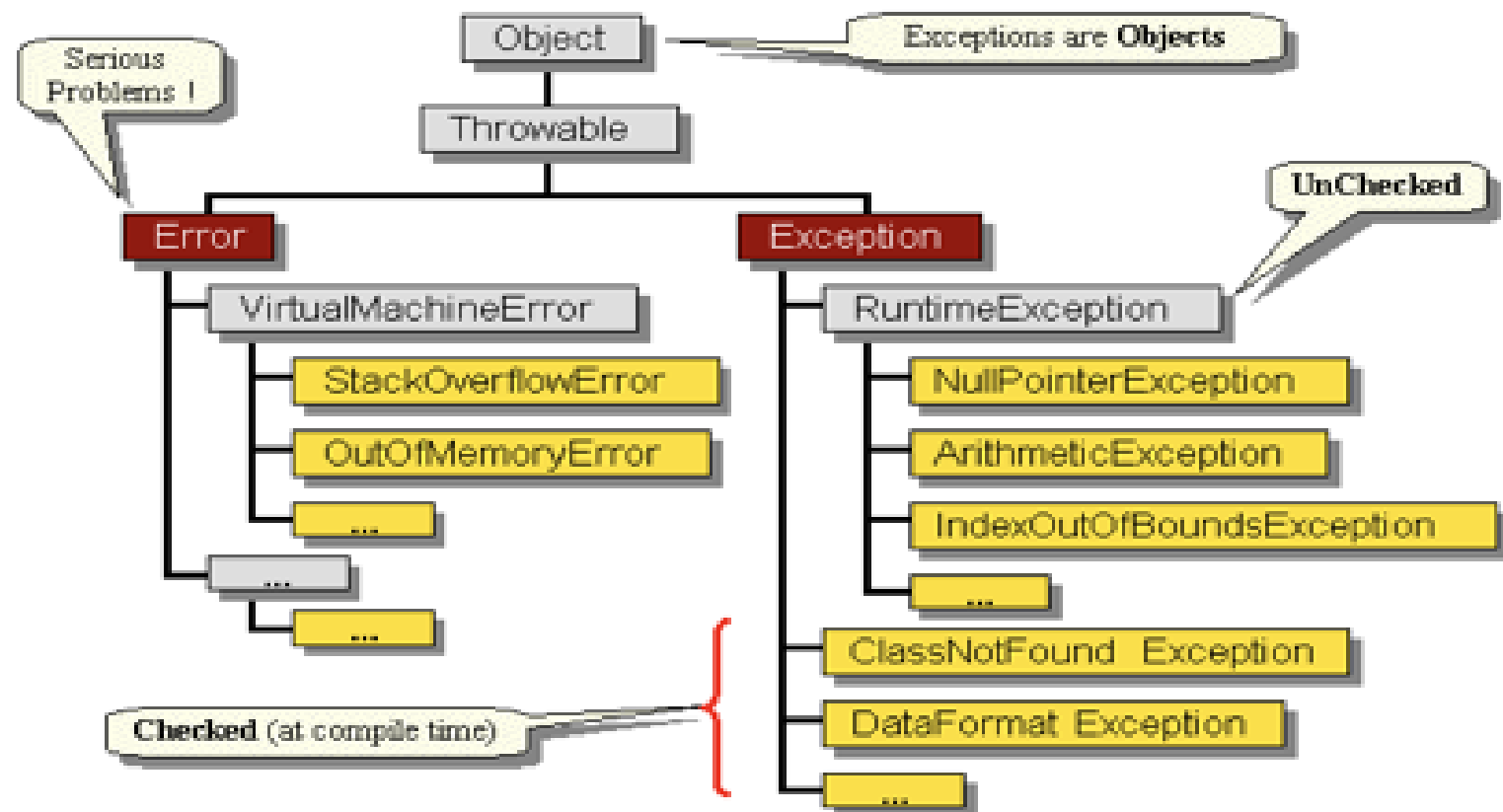
TRATAMENTO DE EXCEÇÕES

- 
- Two sets of three overlapping squares, one in the top-left corner and one in the top-right corner, all with black outlines.
- Uma **exceção** é um evento inesperado que ocorre durante a execução do programa e interrompe o fluxo normal.

Exemplos comuns:

- Divisão por zero.
 - Acesso a índice inexistente em um array.
 - Erro ao abrir um arquivo que não existe.
 - Entrada inválida do usuário.
- 
- A solid black horizontal bar at the bottom of the slide, starting from the left edge and extending across most of the width.

HIERARQUIA DAS EXCEÇÕES



EXCEÇÕES

CHECKED EXCEPTIONS

As checked exceptions, são exceções já previstas pelo compilador e devem ser tratadas ou declaradas.

As classes filhas de Exception são do tipo checked, exceto pelas subclasses de
`java.lang.RuntimeException`

UNCHECKED EXCEPTIONS

As unchecked exceptions (ou exceções não verificadas) são erros que podem acontecer em tempo de execução (runtime) e não precisam ser obrigatoriamente tratadas no código.

TABELA COMPARATIVA

Característica	Checked Exceptions	Unchecked Exceptions
Quando ocorrem	Durante a compilação	Durante a execução (runtime)
Obrigatoriedade de tratar	Obrigatório usar try/catch ou throws	Não obrigatório
Herdam de	Exception (não RuntimeException)	RuntimeException
Exemplos	IOException, SQLException	NullPointerException, ArithmeticException
Atenção:	Você deve se preparar, pois o erro é previsível	Você só descobre quando o erro acontece



TRATANDO AS EXCEÇÕES



Uma maneira de tentar contornar esses imprevistos é realizar o tratamento dos locais no código que podem vir a lançar possíveis exceções, como por exemplo, campo de consulta a [banco de dados](#), locais em que há divisões, consulta a arquivos de propriedades ou arquivos dentro do próprio computador.

Para **tratar as exceções em Java** são utilizados os comandos try e catch.

TRY...CATCH

Onde:

```
try
{
    //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_excecao_1 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_2 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_n e)
{
    //ação a ser tomada
}
```

try{ ... } - Neste bloco são introduzidas todas as linhas de código que podem vir a lançar uma exceção.

catch(tipo_excessao e) { ... } - Neste bloco é descrita a ação que ocorrerá quando a exceção for capturada.

EXEMPLOS

1. `ArithmeticException`

```
public class ExemploTryCatch {
    public static void main(String[] args) {
        try {
            int num = 10 / 0; // causa ArithmeticException
            System.out.println("Resultado: " + num);
        } catch (ArithmeticException e) {
            System.out.println("Erro: divisão por zero não
permitida!");
        }
    }
}
```



EXEMPLOS



1. `ArithmeticException`

```
public class ExemploTryCatch {  
    public static void main(String[] args) {  
        try {  
            int num = 10 / 0; // causa ArithmeticException  
            System.out.println("Resultado: " + num);  
        } catch (ArithmeticException e) {  
            System.out.println("Erro: divisão por zero não  
permitida!");  
        }  
    }  
}
```



EXERCÍCIO



1. `NumberFormatException`

Abra o projeto criado ddd-exception e no pacote models crie a classe `Conversor` cotendo um atributo privado **valor** do tipo **`String`**, criar os getters e setter e um método **`converter()`** que converte o valor digitado para inteiro retornando o mesmo.

Criar a classe `TestaConversor` no pacote test com o método `main`, instanciar a classe `Conversor` e atribuir o valor "12", chamar o método `converter` e exibir o resultado.



EXERCÍCIO



1. `NumberFormatException`

Agora altere o valor "12" para "12a" e observe o que acontece.

Sua tarefa é corrigir este problema.



CORREÇÃO



1. `NumberFormatException`

- Para corrigir este erro podemos lançar o `try...catch`, verificando o erro podemos identificar o tipo de exceção:

```
try {  
    Conversor c1 = new Conversor();  
    c1.setValor("12s");  
    c1.converter();  
} catch (NumberFormatException e) {  
    System.out.println("Valor informado não é um número!");  
}
```

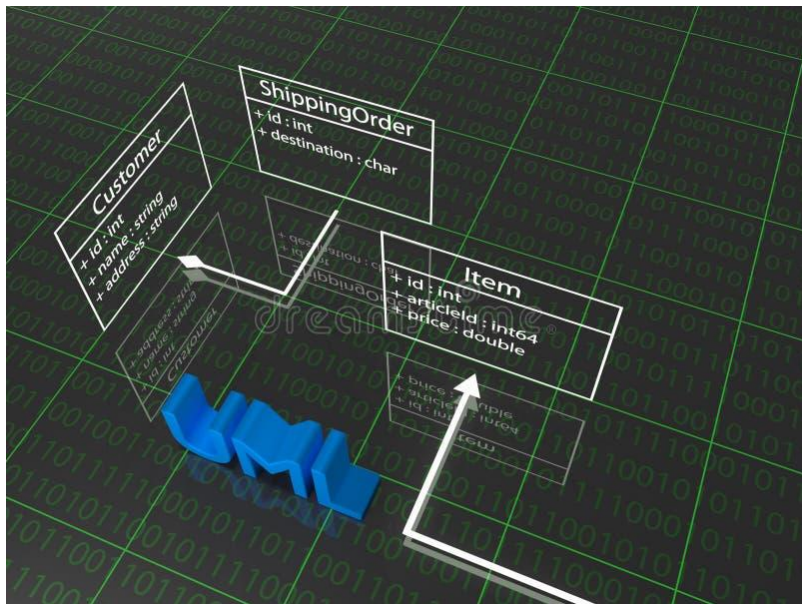


FINALLY



O bloco **finally** sempre é executado, independentemente de ocorrer ou não uma exceção

```
try {  
    String texto = null;  
    System.out.println(texto.length());  
} catch (NullPointerException e) {  
    System.out.println("Objeto nulo!");  
} finally {  
    System.out.println("Finalizando execução...");  
}
```



02
THROW



O **throw** é usado para lançar uma exceção manualmente no seu código.

- Ele interrompe o fluxo normal do programa.
- É como se você dissesse: "Aqui aconteceu um problema, e eu quero avisar disso".

Sua sintaxe é:

```
throw new TipoDaExcecao("mensagem de erro");
```

Onde:

- **throw**: palavra-chave que lança a exceção.
 - **new TipoDaExcecao**: cria um objeto da exceção.
 - **"mensagem de erro"**: texto explicativo para facilitar o diagnóstico.
-



EXEMPLOS



```
public class ExemploThrow {  
    public static void main(String[] args) {  
        verificarIdade(15);  
        System.out.println("Cadastro realizado!");  
    }  
  
    public static void verificarIdade(int idade) {  
        if (idade < 18) {  
            throw new IllegalArgumentException("Idade mínima é  
18 anos!");  
        }  
        System.out.println("Idade válida!");  
    }  
}
```



ENTENDENDO O QUE ACONTECE ...



- Se a idade for **menor que 18**, o throw cria e lança uma **IllegalArgumentException**.
 - O programa **para imediatamente** naquele ponto.
 - Se a idade for válida, o fluxo continua normalmente.
-

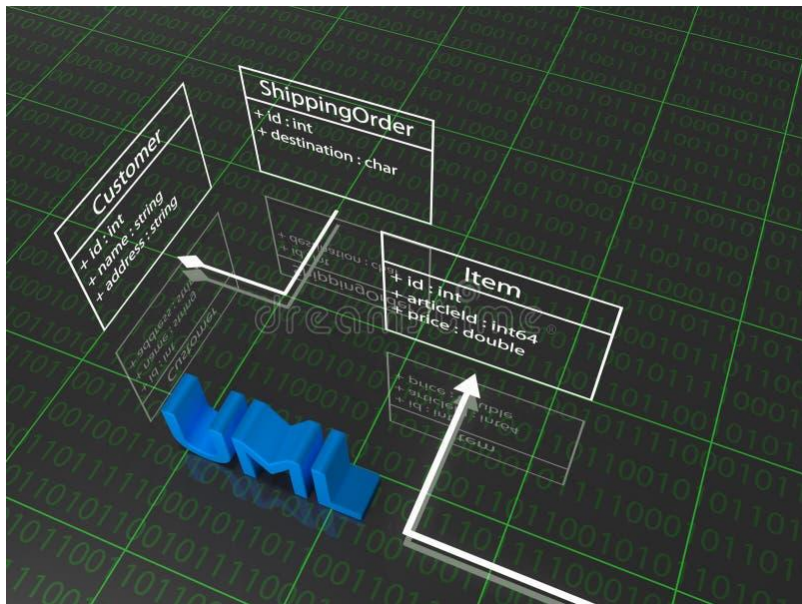


DIFERENÇA ENTRE THROW E THROWS



- throw: **lança a exceção** naquele exato momento (dentro do método).
- throws: **declara que o método pode lançar exceções**, deixando para quem chamar o método decidir se vai tratar.

```
public void lerArquivo(String nomeArquivo) throws IOException {  
    FileReader fr = new FileReader(nomeArquivo);  
    // pode lançar IOException  
}
```



03

CRIANDO SUAS PRÓPRIAS EXCEÇÕES



EXCEÇÕES PERSONALIZADAS



Você pode criar suas próprias exceções para deixar o código mais claro e específico. Para isso, criamos uma classe que herda de `Exception` ou `RuntimeException`.

```
3 public class MinhaExcecao extends Exception{  
4  
5  
6     private static final long serialVersionUID = 1L;  
7     public MinhaExcecao(String texto) {  
8         super(texto);  
9     }  
10  
11 }
```



EXEMPLO



Vamos criar a classe SaldoInsuficienteException

```
// Criando a exceção personalizada
public class SaldoInsuficienteException extends RuntimeException {

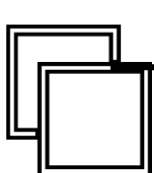
    public SaldoInsuficienteException(String mensagem) {
        super(mensagem); // passa a mensagem para a classe mãe
    }
}
```



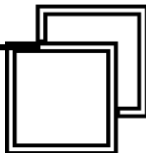
Vamos criar a classe ContaBancaria



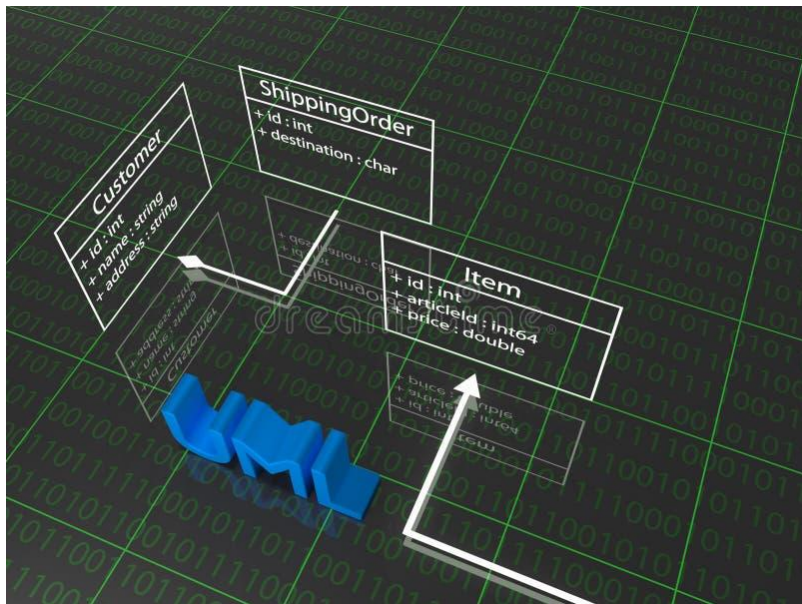
```
public class ContaBancaria {  
    private double saldo;  
  
    public ContaBancaria(double saldoInicial) {  
        this.saldo = saldoInicial;  
    }  
  
    public void sacar(double valor) {  
        if (valor > saldo) {  
            // Lança a exceção personalizada  
            throw new SaldoInsuficienteException("Saldo insuficiente para sacar  
R$" + valor);  
        }  
        saldo -= valor;  
        System.out.println("Saque realizado. Saldo atual: R$" + saldo);  
    }  
}
```



Agora vamos testar, criar a classe TesteConta com o método main



```
public class TesteConta {  
    public static void main(String[] args) {  
        ContaBancaria conta = new ContaBancaria(500.0);  
  
        try {  
            conta.sacar(800.0); // Tentando sacar mais do que tem  
        } catch (SaldoInsuficienteException e) {  
            System.out.println("Erro: " + e.getMessage());  
        }  
  
        System.out.println("Programa finalizado!");  
    }  
}
```



04

EXERCÍCIOS

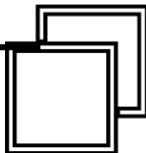
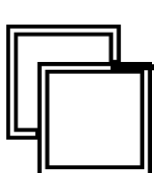


Exercício 1 – Divisão segura

Crie um programa que:

1. Peça dois números ao usuário.
2. Divida o primeiro pelo segundo.
3. Trate a exceção de **divisão por zero** e **entrada inválida**.

Desafio: continue pedindo os números até o usuário digitar valores válidos.



Exercício 2 – Array

Crie um método `buscarElemento(int[] array, int posicao)` que:

Retorne o elemento na posição informada.

Lance uma exceção **`ArrayIndexOutOfBoundsException`** se a posição for inválida.

No main, peça ao usuário a posição e trate a exceção exibindo uma mensagem amigável.

A horizontal line spans the width of the slide. At each end of the line, there is a stack of three squares. The squares are outlined in black and have no fill color. The top square in each stack is slightly offset to the right and top, creating a layered effect.

Exercício 3 – Faça as alterações na classe Conta bancária com exceção personalizada

Adicione na classe ContaBancaria com métodos depositar e sacar.

Se o usuário tentar sacar mais do que tem, lance a exceção personalizada.

No main, trate a exceção e exiba a mensagem ao usuário.

A horizontal line spans the width of the page. At each end of this line, there is a cluster of three overlapping squares. The squares on the left are positioned at the top-left corner, and the squares on the right are positioned at the top-right corner. All squares are empty and have black outlines.

Exercício 4 – Juntando todos os exercícios

Faça uma classe de teste com um menu interativo com os exercícios anteriores e trate todas as exceções de forma adequada.

Referências



Java 6 Ensino Didático

» Sérgio Furgeri - Editora Érika

Java 2 Aprenda em 21 dias

» Rogers Cadenhead, Laura Lemay - Editora Campus

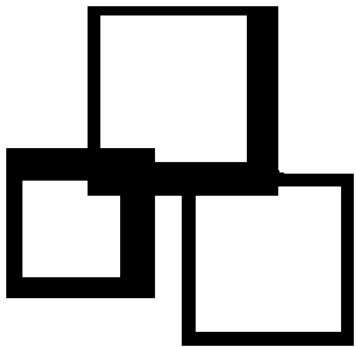
Aprendendo Java

» Niemeyer & Kundsén, Editora Campos

Java e Orientação a Objetos

» Caelum – disponível em

<https://www.caelum.com.br/apostila/apostila-java-orientacao-objetos.pdf>



OBRIGADO

To be continued...

BYE
BYE