



Pulse

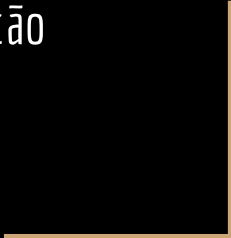
Treinamento Spring
Edenilton Michael de Jesus





Considerações Iniciais

Lógica de programação
Sintaxe geral





JAVA



Considerações Iniciais

JDK

JRE

Estrutura da aplicação Java

Convenções

Garbage Collector

- JAVA DEVELOPMENT KIT
- Bibliotecas de desenvolvimento

Considerações Iniciais

JDK

JRE

Estrutura da aplicação Java

Convenções

Garbage Collector

- JVM - JAVA VIRTUAL MACHINE

Considerações Iniciais

JDK

JRE

Estrutura da aplicação Java

Convenções

Garbage Collector

- Arquivo .java
- Arquivo .class
- Classe Principal

Considerações Iniciais

JDK

JRE

Estrutura da aplicação Java

Convenções

Garbage Collector

- Palavras Reservadas: new, static, etc;
- Nomes das classes iniciam com letra maiúscula;
- Nome de pacotes são sempre com letras minúsculas.

Considerações Iniciais

JDK

JRE

Estrutura da aplicação Java

Convenções

Garbage Collector

- Realiza o trabalho de destruir automaticamente objetos anteriormente instanciados;

Classes

- Atributos
- Construtores
- Métodos
- Parâmetros e argumentos
- Overload e Override
- Acesso a atributos e métodos
- Classes Abstratas
- Classes internas

Atributos

```
public class Empresa  
{  
    int codigo;  
    String descricao;  
    String cnpj;  
    String razaoSocial;  
}
```

Construtores

```
public class Empresa
```

```
{
```

```
    int codigo;
```

```
    String descricao;
```

```
    String cnpj;
```

```
    String razaoSocial;
```

```
    public Empresa(){ }
```

```
    public Empresa(int codigo, String cnpj){
```

```
        this.codigo = codigo;
```

```
        this.cnpj = cnpj;
```

```
    }
```

```
}
```

Métodos

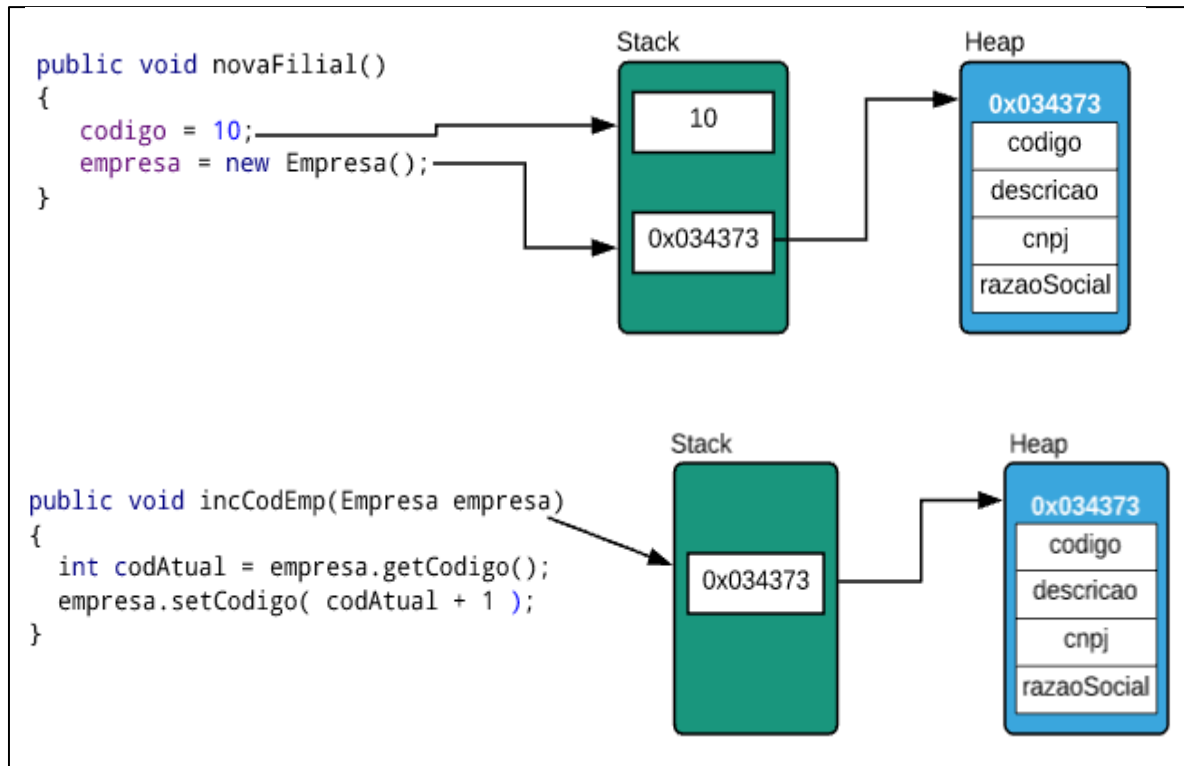
```
public class Empresa
{
    int codigo;
    String descricao;
    String cnpj;
    String razaoSocial;

    public void setDescricao(String descricao){
        this.descricao = descricao;
    }

    public String getDescricao(){
        return descricao;
    }
}
```

Parâmetros e argumentos

- **Parâmetros são sempre passados por valor;**
- Argumentos são sempre passados na chamada da aplicação.



Parâmetros e argumentos

- Parâmetros são sempre passados por referência;
- **Argumentos são sempre passados na chamada da aplicação.**

```
public static void main(String[] args)
{
    //ponto de início da aplicação
}
```

chamada: aplicacao arg1 arg2 argn

Overloading

- Métodos com mesmo nome
- Mas não com a mesma assinatura!

```
public void remover(Empresa empresa)
{
}
public void remover(String cnpj)
{
}
public void remover(int codigo)
{
}
```

Acesso a atributos e métodos

public

private

protected

```
public class Empresa  
{  
    public String cep;  
}
```

- Atributo cep pode ser acessado diretamente.

```
Empresa empresa = new Empresa();  
empresa.cep = "65000000";
```


Acesso a atributos e métodos

public

private

protected

```
public class Empresa  
{  
    private String cnpj;  
}
```

- Atributo cnpj somente pode ser acessado dentro da classe.

```
Empresa empresa = new Empresa();  
empresa.cnpj = "12345678909876"; //erro!
```

Acesso a atributos e métodos

public

private

protected

```
public class Empresa
{
    protected double capitalSocial;
}
```

- Atributo capitalSocial pode ser acessado por uma subclasse de Empresa;
- Ou por uma classe que esteja dentro do mesmo pacote que Empresa..

```
Empresa empresa = new Empresa();
empresa.capitalSocial = 650000.50;
```

Classes Abstratas

- Nunca são instanciadas;
- Reúnem características comuns a suas subclasses;
- Normalmente não possuem construtores;
- Podem possuir métodos concretos.

```
public abstract class Empresa
{
    private int codigo;
    private String descricao;
    private String cnpj;
    private String razaoSocial;

    public abstract void atualizarDados();

    public void limparDescricao(){
        descricao = "";
    }
}
```

Classes Internas

- Utilizadas no contexto interno da classe hospedeira;
- Pode ser usada fora, dependendo do seu modificador de acesso;
- Acessa atributos privados da classe hospedeira.

```
public class Empresa
{
    private int codigo;

    private class C1{
        private int cod;

        public C1(){
            this.cod = codigo;
        }
    }

    public class C2{
        private int cod;
    }

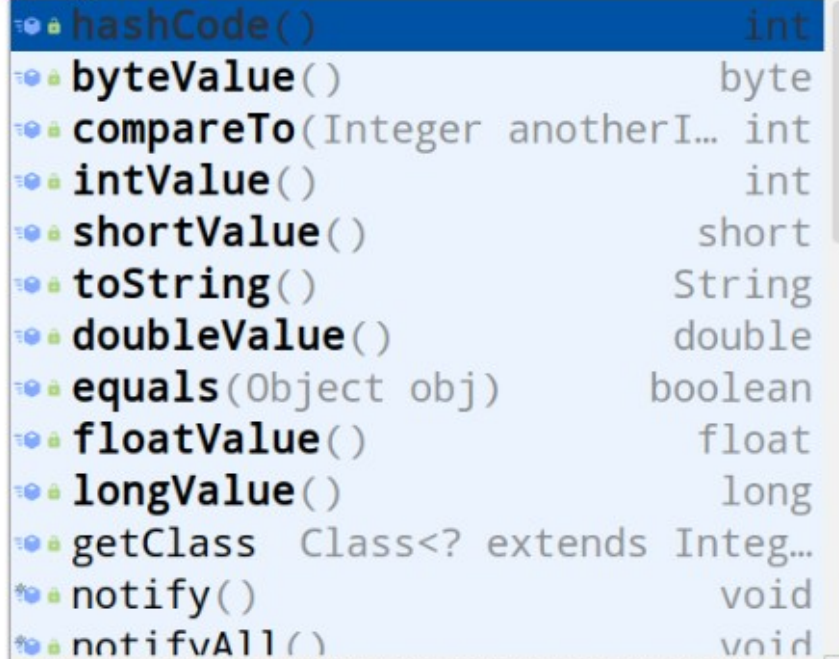
    protected class C3{
        private int cod;
    }
}
```

Tipos de dados

PRIMITIVO	REFERÊNCIA
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
-	String

Tipos de dados

```
public C1(){  
    this.cod = codigo.};  
}  
  
public class C2{  
    private int cod;  
}  
  
protected class C3{  
    private int cod;  
}
```



hashCode()	int
byteValue()	byte
compareTo(Integer anotherI...	int
intValue()	int
shortValue()	short
toString()	String
doubleValue()	double
equals(Object obj)	boolean
floatValue()	float
longValue()	long
getClass Class<? extends Integ...	
notify()	void
notifyAll()	void

Did you know that Quick Definition View (Ctrl+Shift+I) works in completion lookups as well? >>

0-0

Tipos de dados

Números Gigantes
BigInteger
BigDecimal

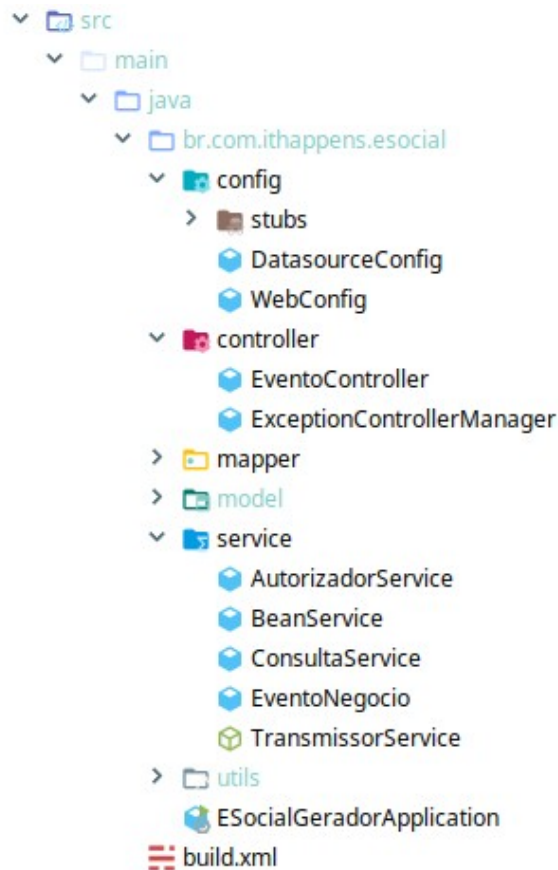
Alguns Operadores:

- BigInteger add(BigInteger other)
- BigInteger subtract(BigInteger other)
- BigInteger multiply(BigInteger other)
- BigInteger divide(BigInteger other)
- BigInteger mod(BigInteger other)

```
BigInteger a = new BigInteger("5948693968573654916751657619646");  
BigInteger b = BigInteger.TEN;  
BigInteger c = a.add(b); // c = a + b  
  
BigInteger d = c.multiply(b.add(BigInteger.valueOf(2))); // d = c * (b+2)
```

Pacotes

- Grupo de classes;
- Pacotes diferentes podem possuir classes com mesma assinatura.



Pacotes

- package;
- import;
- java.io,
java.lang,
java.util.

```
package br.com.ithappens.esocial.model.banco;
```

```
import br.com.ithappens.esocial.model.util.Par;
```

```
import java.util.List;
```

```
import java.util.Optional;
```

```
public class Configuracao
```

```
{
```

```
    private int id;
```

```
    private String hashCertificado;
```

```
    private String senhaCertificado;
```

```
    private String cnpjRaiz;
```

```
    private Empresa empresa;
```

```
}
```

Interfaces

Não é uma classe, mas um conjunto de requisitos para as classes que desejam estar em conformidade com a interface.

Normalmente, o fornecedor de alguns serviços afirma: “Se a sua aula estiver em conformidade com uma interface específica, então irei realizar o serviço.” Vejamos um exemplo concreto. O método de classificação da classe Arrays promete classificar uma matriz de objetos, mas sob uma condição: Os objetos devem pertencer a classes que implementam a interface ***Comparable***.

Interfaces

- Agrupamento de métodos;
- Os métodos são genéricos e podem ser implementados por qualquer classe, independente da hierarquia;
- Uma mesma classe pode implementar métodos de várias interfaces diferentes;
- Interfaces podem generalizar outras interfaces, inclusive com herança múltipla.

Interfaces

- Podem possuir atributos;
- Atributos declarados servem de constantes para a implementação concreta da interface.

```
public interface ConfiguracaoMapper
{
    String ATRIBUTO1 = "valor";
    Integer ATRIBUTO2 = 10;

    List<Configuracao> configuracoes();
    Configuracao getByEmpresa(Empresa empresa);
}
```

Interfaces

- Podem possuir atributos;
- Atributos declarados servem de constantes para a implementação concreta da interface.

```
public interface ConfiguracaoMapper extends Interface1, Interface2, interfaceN
{
    String ATRIBUTO1 = "valor";
    Integer ATRIBUTO2 = 10;

    List<Configuracao> configuracoes();
    Configuracao getByEmpresa(Empresa empresa);
}
```

Interfaces

- Classes que estabelecem um contrato com uma interface, tem obrigação de sobrescrever a implementação dos métodos declarados;

```
public interface INotaFiscal{
    void imprimir();
    boolean enviar(NotaFiscal notaFiscal);
}

public class ServicoNotaV3 implements INotaFiscal
{

    @Override
    public void imprimir()
    {
    }

    @Override
    public void enviar(NotaFiscal notaFiscal)
    {
    }
}
```

Enums

```
public enum EnumOperacoes
{
    INCLUSAO,
    EXCLUSAO,
    ALTERACAO;
}
```

```
public enum EnumOperacoes
{
    INCLUSAO("INCLUSAO"),
    EXCLUSAO("EXCLUSAO"),
    ALTERACAO("ALTERACAO");

    private String operacao;

    EnumOperacoes(String operacao){
        this.operacao = operacao;
    }
}
```

Orientação a objetos

Abstração

Herança

Encapsulamento

Polimorfismo

Herança

```
public class Conta
{
    protected double saldo;
    protected Cliente cliente;

    public boolean sacar(double valor){}
    public boolean depositar(double valor){}
}
```

```
public class ContaCorrente extends Conta
{
    private double taxaServico;
}
```

```
public class ContaPoupanca extends Conta
{
    private double taxaRendimento;
}
```

```
ContaCorrente cc = new ContaCorrente();
cc.sacar(10.0);
cc.getSaldo();
```

```
ContaPoupanca cp = new ContaPoupanca();
cp.sacar(30.0);
cp.getSaldo();
```

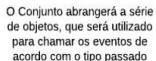
Herança - Classes Abstratas

- São classes base utilizadas como superclasses;
- Não podem ser instanciadas;
- Devem atender a um contexto bem definido e restrito;
- Infelizmente, há um grande problema em usar uma classe base abstrata para expressar uma propriedade genérica, porque uma classe pode estender apenas uma única classe.

```
abstract class Comparable // why not?  
{  
    public abstract int compareTo(Object other);  
}
```

```
public interface Comparable<T>  
{  
    int compareTo(T other);  
}
```

ESOCIAL
WEBSERVICE
RH



S2240

S2221

Encapsulamento

- Ocultar propriedades e comportamentos;
- Chamada **empresa.metodoQualquer()** gera erro;
- Chamada **empresa.metodoQualquer("valor")** não gera erro.

```
public class Empresa
{
    private Integer codigo;
    private String descricao;
    private String cnpj;
    private String razaoSocial;

    public void metodoQualquer(String parametro){
    }

    private void metodoQualquer(){
    }
}
```

Polimorfismo

- Objetos apresentam comportamentos distintos para o mesmo tipo de ação;

```
public interface IEntradaSaida{
    void salvar(Object conteudo);
    Object ler();
}

public class Classe1 implements IEntradaSaida
{
    // Implementacao dos metodos
}

public class Classe2 implements IEntradaSaida
{
    // Implementacao dos metodos
}

IEntradaSaida es = new Classe1();
es.ler();
es = new Classe2();
es.ler();
```

Generics

- Significa escrever código que pode ser reutilizado para objetos de muitos tipos diferentes.
- Por exemplo, não necessariamente se necessita implementar classes separadas para armazenar um texto ou um número.

Generics

```
public interface CRUD<T>
{
    public boolean salvar(T objeto);
    public boolean atualizar(T objeto);
    public boolean excluir(T objeto);
    public T ler(Integer codigo);
}

public class empresaDao implements CRUD<Empresa>
{
}

public class FilialDao implements CRUD<Filial>
{
}
```

- Métodos genéricos:

```
public <T> T getValor(Object base, Class<T> clazz)
{
}
```

- Atributos genéricos:

```
public class Fila<T>
{
    private int tamanho;
    private T elementos[];

    public boolean inserir(T objeto);
    public boolean remover(T objeto);
}
```

Wildcards

Descrição	Declaração	Recomendação
Upper Bound	? extends Class	Operações De Leitura
Lower Bound	? super Class	Operações de Escrita
UnBound	?	Qualquer Situação

Wildcards

```
class C0
{
}
class C1 extends C0
{
}
class C2 extends C1
{
}
class C3 extends C1
{
}
```

```
public void upperBound(List<? extends C1> lista){
    lista.forEach( System.out::println );
}
public void lowerBound(List<? super C1> lista){
    lista.add( new C1() );
    lista.add( new C0() );
}
public void unBound(List<?> lista){
    lista.forEach(System.out::println);
}

upperBound(Arrays.asList( new C2(), new C3() ));
lowerBound(new ArrayList<C0>());
unBound(Arrays.asList( new C2(), new C3() ));
```

Tratamento de Exceções

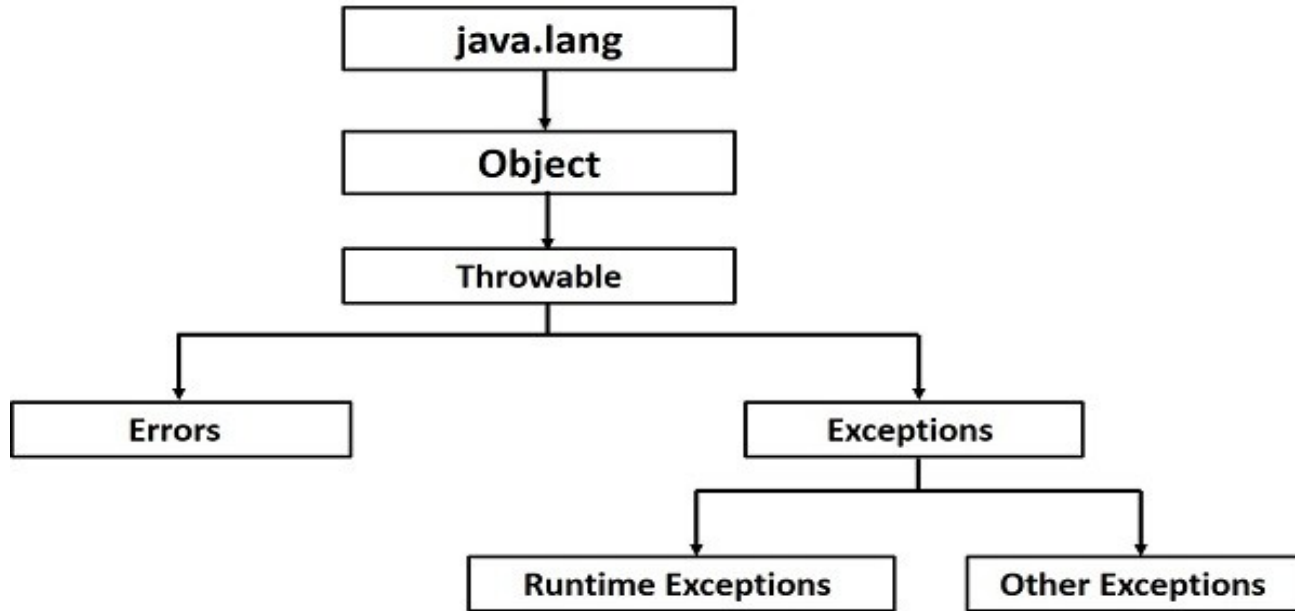
Arquitetura de exceções

Try/Catch/Finally

Lançamento de exceções

Exceções customizadas

Arquitetura de exceções



Try, Catch e Finally

- Try: Tentativa de execução;
- Catch: Caso ocorra alguma exceção, esse bloco a captura e é executado;
- Finally: Independente de haver exceção ou não, esse bloco sempre será executado.

```
public void executaSQL(){
    Conexao conexao = null;
    try{
        conexao = new Conexao();
        conexao.executaSQL( SQL );
    }catch( SQLException ex){
        ex.printStackTrace();
        //outras verificações aqui
    }finally {
        conexao.close();
    }
}
```

Try, Catch e Finally

- Try: Tentativa de execução;
- Catch: Caso ocorra alguma exceção, esse bloco captura e é executado;
- Finally: Independente de haver exceção ou não, esse bloco sempre será executado.

```
public void executaSQL(){
    Conexao conexao = null;
    try{
        conexao = new Conexao();
        conexao.executaSQL( SQL );
    }catch( SQLException | IOException ex){
        ex.printStackTrace();
        //outras verificações aqui
    }finally {
        conexao.close();
    }
}
```

Try, Catch e Finally

- Try: Tentativa de execução;
- Catch: Caso ocorra alguma exceção, esse bloco captura e é executado;
- Finally: Independente de haver exceção ou não, esse bloco sempre será executado.

```
public void executaSQL(){
    Conexao conexao = null;
    try{
        conexao = new Conexao();
        conexao.executaSQL( SQL );
    }catch( SQLException ex){
        ex.printStackTrace();
        //outras verificações aqui
    }catch( IOException ioex){
        ioex.printStackTrace();
        //outras verificações aqui
    }
    finally {
        conexao.close();
    }
}
```

Lançamento de exceções

- Throws;
- Throw.

```
public String lerArquivoTexto(String caminho) throws IOException {  
    File arquivo = new File(caminho);  
    if( !arquivo.exists() ){  
        throw new IOException();  
    }  
    String conteudoArquivo = instrucoesLeituraArquivi();  
    return conteudoArquivo;  
}
```

Exceções customizadas

- Exceções customizadas de acordo com a regra de negócio;
- Facilita o debug.

```
class EmpresaNotFoundException extends Exception{  
    public EmpresaNotFoundException(String message){  
        super(message);  
    }  
}  
  
public Empresa buscarEmpresa(Integer codigo) throws EmpresaNotFoundException  
{  
    EmpresaDao empresaDao = new EmpresaDao();  
    Empresa empresa = empresaDao.buscar(codigo);  
    if( empresa == null ){  
        throw new EmpresaNotFoundException("Empresa não encontrada");  
    }  
    return empresa;  
}
```

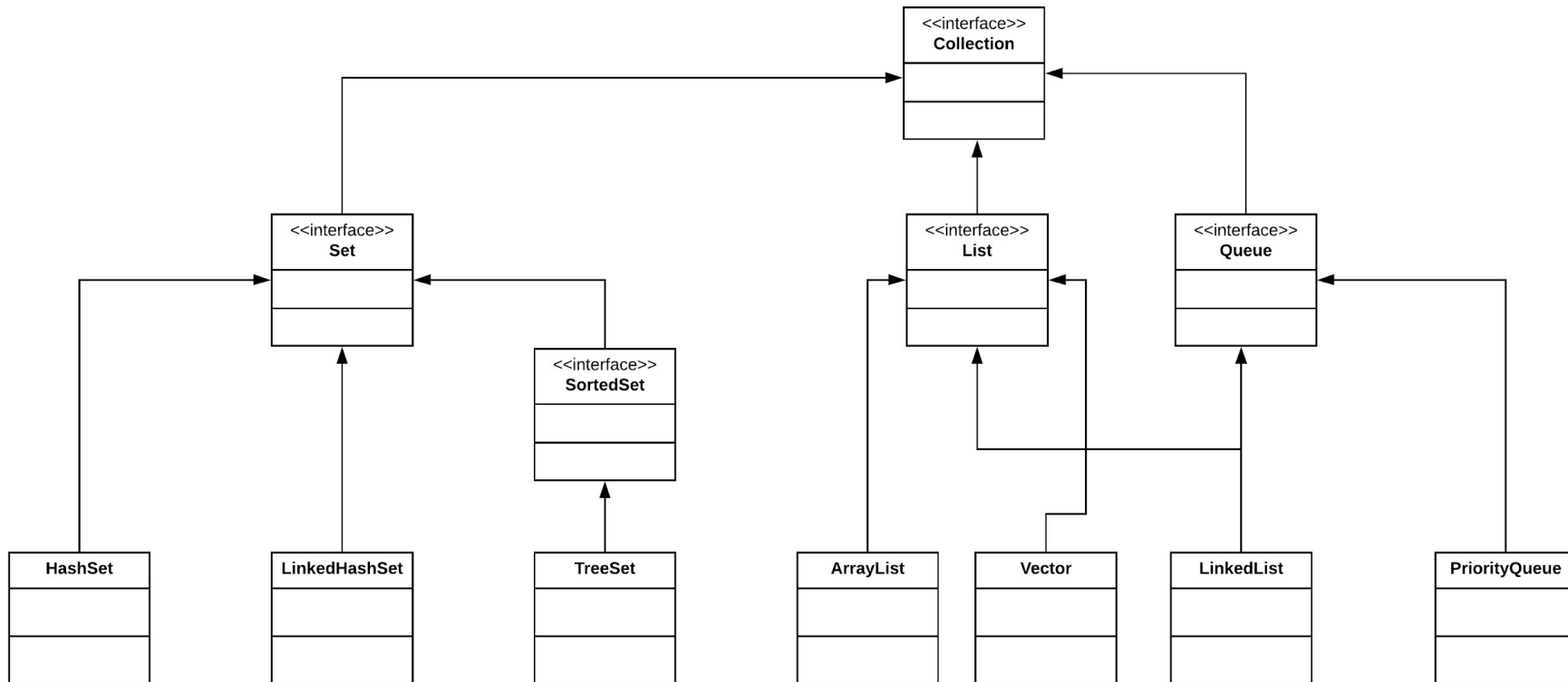

Exceções - Dicas

- Não utilize sempre a classe Exception;
- Não crie muitas exceções específicas;
- Utilize log para exibir o que ocorreu de exceção, porém foque em um ponto específico;
- Checked Vs unchecked exceptions;
- Se tentar fazer algo que possa gerar exceção(try), sempre trate a exceção(catch);
- Se você lança um exceção, trate-a o mais tarde possível;
- Encadeie o stack trace ao lançar uma exceção dentro de um bloco catch;

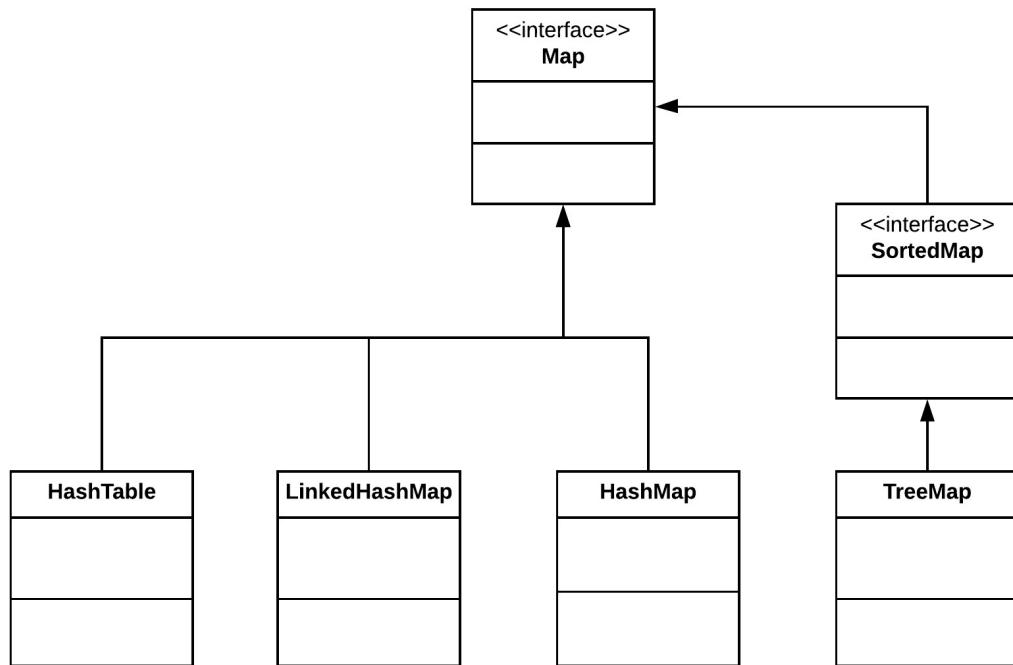
Coleções

- Conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade;
- **Interfaces:** tipos abstratos que representam as coleções;
- **Implementações:** são as implementações concretas das interfaces;
- **Algoritmos:** são os métodos que realizam as operações sobre os objetos das coleções

Coleções



Mapas



Coleções e Mapas

Interfaces	Espalhamento	Array Dinâmico	Árvore	Lista Ligada	Espalhamento + Lista Ligada
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

Coleções

```
public void exemplo() {  
    List<Produto> produtos = new ArrayList<>();  
    produtos.add( new Produto(1L, "Notebook"));  
    produtos.add( new Produto(2L, "Smartphone"));  
    produtos.size();  
    produtos.get(0);  
    produtos.contains( new Produto(1L, "Notebook") );  
}
```

Coleções

- Métodos equals e hashCode são importantes;
- Usados para comparações na regra de negócio;
- Usados pelas coleções em seus algoritmos internos, como busca, armazenamento.

```
public class Produto {  
    private Long codigo;  
    private String descricao;  
  
    public boolean equals(Object obj) {  
        if( this == obj ){ return true; }  
        if(obj==null || !getClass().equals(obj.getClass())){  
            return false;  
        }  
        Produto objProduto = (Produto) obj;  
        return getCodigo().equals(objProduto.getCodigo());  
    }  
  
    public int hashCode() {  
        return Objects.hash(codigo);  
    }  
}
```

Receita de implementação do método equals

Nomeie o parâmetro explícito ***otherObject***, pois mais tarde, você precisará convertê-lo em outra variável que deve ser chamada de ***other***.

```
public boolean equals(Object otherObject) {  
    Message other = (Message) otherObject;  
}
```


Receita de implementação do método equals

Teste se o seu objeto (**this**) é idêntico a **otherObject**.

```
public boolean equals(Object otherObject) {  
    if (this == otherObject) {  
        return true;  
    }  
}
```

Receita de implementação do método equals

Testa se ***otherObject*** é nulo e retorna falso se for. Este teste é obrigatório.

```
public boolean equals(Object otherObject) {  
    if (otherObject == null) {  
        return false;  
    }  
}
```

Receita de implementação do método equals

Compare as classes deste objeto (**this**) e de **otherObject**. Se a semântica do equals pode mudar nas subclasses, use o teste com o método *getClass*.

Se a mesma semântica for válida para todas as subclasses, você pode usar uma **instanceof** no teste.

```
public boolean equals(Object otherObject) {  
    if (getClass() != otherObject.getClass()) {  
        return false;  
    }  
}
```

```
public boolean equals(Object otherObject) {  
    if (!(otherObject instanceof ClassName)) {  
        return false;  
    }  
}
```

Receita de implementação do método equals

Transmita ***otherObject*** para uma variável do seu tipo de classe.

```
public boolean equals(Object otherObject) {  
    ClassName other = (ClassName) otherObject;  
}
```

Receita de implementação do método equals

Compare os campos, conforme exigido por sua noção de igualdade. Use `==` para campos de tipo primitivo, ***Objects.equals*** para campos de objeto. Retorna verdadeiro se todos os campos corresponderem, caso contrário, retorna falso.

```
public boolean equals(Object otherObject) {  
    return field1 == other.field1 &&  
        Objects.equals(field2, other.field2);  
}
```

Observação de implementação do método equals

Muito cuidado para não implementar o método equals com o parâmetro diretamente do tipo ao qual a classe pertence, pois fazendo isso, você não estará sobrescrevendo o método original que se é herdado da classe Object, podendo ocasionar falha na sua utilização, como por exemplo, em coleções.

```
public class NFCe
{
    // ERRADO!
    public boolean equals(NFCe other){
        //...Implementação aqui
    }
}

public class NFCe
{
    // CORRETO!
    public boolean equals(Object otherObject){
        //...Implementação aqui
    }
}
```

HashCode

- Um código hash é um número inteiro derivado de um objeto. Os códigos hash devem ser embaralhados, então se `x` e `y` forem dois objetos distintos, deve haver uma alta probabilidade de que `x.hashCode()` e `y.hashCode()` sejam diferentes;
- O método `hashCode` é definido na classe `Object`, então, todo objeto possui um código hash padrão;
- Se você redefinir o método `equals`, você também precisará redefinir o método `hashCode` para objetos que os usuários podem inserir em uma tabela hash;
- Suas definições de `equals` e `hashCode` devem ser compatíveis: Se `x.equals(y)` for verdadeiro, então `x.hashCode()` deve retornar o mesmo valor que `y.hashCode()`

HashCode

O método hashCode deve retornar um inteiro (que pode ser negativo). Basta combinar os códigos hash dos campos de instância para que os códigos hash para objetos diferentes sejam amplamente dispersos.

```
public class Produto
{
    public int hashCode()
    {
        return 7 * codigo.hashCode() +
            11 * Objects.hashCode(ncm);
        //return Objects.hash(codigo, ncm);
    }
}
```




JAVA 8



Considerações Iniciais

- Programação Funcional
- Lambdas
- Streams
- Datas

Interface Methods

- Default Methods
- Static Methods;
- Abstract methods.

```
public interface I {  
    static void printStatic(){  
        System.out.println("print static");  
    }  
  
    default void printDefault(){  
        System.out.println("print default");  
    }  
  
    abstract void printAbstract();  
}  
  
public class C implements I {  
    @Override  
    public void printAbstract() {  
        System.out.println("Implementado");  
    }  
}
```

Interfaces Funcionais

- Contém apenas um método abstrato;
- Você especifica o que quer alcançar em uma tarefa, mas não como alcançar isso;
- Imutabilidade

```
public interface I {  
    default void printDefault(){  
        System.out.println("print default");  
    }  
  
    abstract void printAbstract();  
}  
  
public interface I2 {  
    abstract void printAbstract();  
    abstract void printAbstract2();  
}  
  
public interface I3 {  
    abstract void printAbstract();  
}
```

Interfaces Funcionais

Interface	Descrição
<code>BinaryOperator<T></code>	Contém o método <code>apply</code> , que recebe dois argumentos <code>T</code> , realiza uma operação neles (como um cálculo) e retorna um valor do tipo <code>T</code> . Veremos vários exemplos de <code>BinaryOperators</code> a partir da Seção 17.3.
<code>Consumer<T></code>	Contém o método <code>accept</code> , que recebe um argumento <code>T</code> e retorna <code>void</code> . Realiza uma tarefa com o argumento <code>T</code> , como gerar uma saída do objeto, chamar um método do objeto etc. Veremos vários exemplos de <code>Consumers</code> a partir da Seção 17.3.
<code>Function<T, R></code>	Contém o método <code>apply</code> , que recebe um argumento <code>T</code> e retorna um valor do tipo <code>R</code> . Chama um método no argumento <code>T</code> e retorna o resultado desse método. Veremos vários exemplos de <code>Functions</code> a partir da Seção 17.5.
<code>Predicate<T></code>	Contém o método <code>test</code> , que recebe um argumento <code>T</code> e retorna um <code>boolean</code> . Testa se o argumento <code>T</code> atende uma condição. Veremos vários exemplos de <code>Predicates</code> a partir da Seção 17.3.
<code>Supplier<T></code>	Contém o método <code>get</code> , que não recebe argumentos e produz um valor do tipo <code>T</code> . Muitas vezes usado para criar um objeto de coleção em que os resultados de uma operação de fluxo são inseridos. Veremos vários exemplos de <code>Suppliers</code> a partir da Seção 17.7.
<code>UnaryOperator<T></code>	Contém o método <code>get</code> que não recebe argumentos e retorna um valor do tipo <code>T</code> . Veremos vários exemplos de <code>UnaryOperators</code> a partir da Seção 17.3.

Expressões Lambda

Uma expressão lambda é um bloco de código que você pode passar para que possa ser executado posteriormente, uma ou várias vezes.

```
Comparable<Produto> compProdutoPreco = (p) -> {  
    if(p.getPreco() > 2.50){  
        return 1;  
    }else if( p.getPreco() < 2.50 ){  
        return -1;  
    }else{  
        return 0;  
    }  
}
```

Expressões Lambda

Sintaxe: (listaDeParâmetros) ->
{instruções}

```
(int x, int y) -> {return x + y;;}
```

```
(x, y) -> {return x + y;;}
```

```
(x, y) -> x + y;
```

```
() -> System.out.println("lambdas!");
```

Expressões Lambda

```
public interface I {  
    default void printDefault(){  
        System.out.println("print default");  
    }  
  
    abstract void printAbstract();  
}
```

```
I i = new I() {  
    @Override  
    public void printAbstract() {  
        System.out.println("ok");  
    }  
};  
  
I i2 = () -> {  
    System.out.println("ok");  
};  
  
I i3 = () -> System.out.println("ok");
```


Streams

- **Sequência de elementos;**
- Fonte de dados;
- Operações de agregação;
- Estrutura de processo;
- Iteração interna.

Uma stream oferece uma interface para um conjunto de valores sequenciais de um tipo de elemento particular. Apesar disso, as streams não armazenam elementos; estes são calculados sob demanda.

Streams

- Sequência de elementos;
- **Fonte de dados;**
- Operações de agregação;
- Estrutura de processo;
- Iteração interna.

As streams tomam seu insumo de uma fonte de dados, como coleções, matrizes ou recursos de E/S.

Streams

- Sequência de elementos;
- Fonte de dados;
- **Operações de agregação;**
- Estrutura de processo;
- Iteração interna.

As streams suportam operações do tipo SQL e operações comuns à maioria das linguagens de programação funcionais, como filter, map, reduce, find, match e sorted, entre outras.

Streams

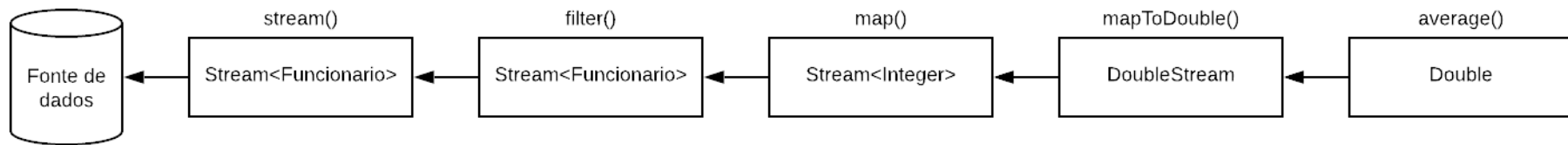
- Sequência de elementos;
- Fonte de dados;
- Operações de agregação;
- **Estrutura de processo;**
- Iteração interna.

Muitas operações de stream retornam outra stream. Assim, é possível encadear operações para formar um processo mais abrangente. Isto, por sua vez, permite algumas otimizações, por exemplo mediante as noções de "preguiça" (*laziness*) e "corte de circuitos" (*short-circuiting*).

Streams

- Sequência de elementos;
- Fonte de dados;
- Operações de agregação;
- **Estrutura de processo;**
- Iteração interna.

Muitas operações de stream retornam outra stream. Assim, é possível encadear operações para formar um processo mais abrangente. Isto, por sua vez, permite algumas otimizações, por exemplo mediante as noções de "preguiça" (*laziness*) e "corte de circuitos" (*short-circuiting*).



Streams

- Sequência de elementos;
- Fonte de dados;
- Operações de agregação;
- Estrutura de processo;
- **Iteração interna.**

Diferentemente do trabalho com coleções, em que a iteração é explícita (iteração externa), as operações da stream realizam uma iteração por trás dos bastidores.

Streams

- Fluxos são objetos das classes que implementam a interface Stream;
- Juntamente com lambdas, fluxos permitem realizar tarefas sobre coleções de elementos.
- Operações terminais e intermediárias

Streams

Operações Stream intermediárias

<code>filter</code>	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
<code>distinct</code>	Resulta em um fluxo que contém somente os elementos únicos.
<code>limit</code>	Resulta em um fluxo com o número especificado de elementos a partir do início do fluxo original.
<code>map</code>	Resulta em um fluxo em que cada elemento do fluxo original é mapeado para um novo valor (possivelmente de um tipo diferente) — por exemplo, mapear valores numéricos para as raízes quadradas dos valores numéricos. O novo fluxo tem o mesmo número de elementos que o fluxo original.
<code>sorted</code>	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.

Streams

Operações Stream terminais

`forEach` Realiza o processamento em cada elemento em um fluxo (por exemplo, exibir cada elemento).

Operações de redução — *recebem todos os valores no fluxo e retornam um único valor*

`average` Calcula a *média* dos elementos em um fluxo numérico.

`count` Retorna o *número de elementos* no fluxo.

`max` Localiza o *maior* valor em um fluxo numérico.

`min` Localiza o *menor* valor em um fluxo numérico.

`reduce` Reduz os elementos de uma coleção a um *único valor* usando uma função de acumulação associativa (por exemplo, uma lambda que adiciona dois elementos).

Operações de redução mutáveis — *criam um contêiner (como uma coleção ou um `StringBuilder`)*

`collect` Cria uma *nova coleção* dos elementos contendo os resultados das operações anteriores do fluxo.

`toArray` Cria um *array* contendo os resultados das operações anteriores do fluxo.

Streams

Operações de pesquisa

<code>findFirst</code>	Localiza o <i>primeiro</i> elemento no fluxo com base nas operações intermediárias anteriores; termina imediatamente o processamento do pipeline do fluxo depois que esse elemento é encontrado.
<code>findAny</code>	Localiza <i>qualquer</i> elemento no fluxo com base nas operações intermediárias anteriores; termina imediatamente o processamento do pipeline do fluxo depois que esse elemento é encontrado.
<code>anyMatch</code>	Determina se <i>quaisquer</i> elementos no fluxo correspondem a uma condição especificada; termina imediatamente o processamento do pipeline do fluxo se um elemento corresponde.
<code>allMatch</code>	Determina se <i>todos</i> os elementos no fluxo correspondem a uma condição especificada.

Streams

```
String[] strings = {"Red", "orange", "Yellow",  
"green", "Blue", "indigo", "Violet"};  
// exibe strings originais  
System.out.printf("Original strings: %s%n",  
Arrays.asList(strings));  
// strings em maiúsculas  
System.out.printf("strings in uppercase: %s%n",  
    Arrays.stream(strings)  
        .map(String::toUpperCase)  
        .collect(Collectors.toList()));
```

```
// strings menores que "n" (sem distinção maiúsc/minúsc) em ordem  
crescente  
System.out.printf("strings greater than m sorted ascending: %s%n",  
    Arrays.stream(strings)  
        .filter(s -> s.compareToIgnoreCase("n") < 0)  
        .sorted(String.CASE_INSENSITIVE_ORDER)  
        .collect(Collectors.toList()));  
  
// strings menores que "n" (com distinção maiúsc/minúsc) em ordem  
decrecente  
System.out.printf("strings greater than m sorted descending: %s  
%n",  
    Arrays.stream(strings)  
        .filter(s -> s.compareToIgnoreCase("n") <  
0).sorted(String.CASE_INSENSITIVE_ORDER.reversed())  
        .collect(Collectors.toList()));
```

Streams

```
List<Integer> lista =  
Arrays.asList( 3,2,1,4,7,6,5,8,9,0  
);  
System.out.printf("Soma do dobro  
números pares %d%n",  
    lista.stream()  
        .filter( v -> v % 2 == 0 )  
        .map( v -> v * 2 )  
        .reduce(Integer::sum)  
        .orElse(0);
```

System.out.printf("Retorna uma lista dos
números ímpares da lista original, ordenada
do maior para o menor ");

```
List<Integer> imparesOrdenados = lista  
    .stream()  
        .filter( v -> v % 2 != 0 )  
        .sorted( Collections.reverseOrder() )  
        .collect( Collectors.toList() );  
  
imparesOrdenados.forEach  
(  
    System.out::println  
);
```

Iteração externa e interna

```
List<String> nomes = Arrays.asList( "Treinamento", "ITHappens", "Módulo 01" );
```

// Iteração externa comum

```
for(int i = 0 ; i < nomes.size();i++){  
    System.out.println(nomes.get(i));  
}
```

// Iteração externa com foreach

```
for(String nome : nomes){  
    System.out.println(nome);  
}
```

// Iteração interna com foreach

```
nomes.forEach((nome) -> System.out.println(nome));
```

// Iteração interna com foreach e method reference

```
nomes.forEach(System.out::println);
```

SERVIDORES DE APLICAÇÕES

Apresentação



- API padronizada para acesso a recursos Java EE;
- Uma implementação para o modelo de arquitetura JAVA EE para computação distribuída incluindo acesso e comunicação entre componentes locais e remotos;
- Uma ou mais interfaces de gerenciamento que permitam controle integrado dos recursos disponibilizados a aplicações;
- Wildfly, Glassfish, Tomcat;
- Tomcat: container Web de código aberto, que foi criado para executar aplicações em tecnologias Servlets.

ESTRUTURA DE PROJETO



MAVEN

MAVEN

- Configuração de projeto simples que segue as melhores práticas - obtenha um novo projeto ou módulo iniciado em segundos;
- Uso consistente em todos os projetos - significa que não há tempo de aceleração para novos desenvolvedores entrarem em um projeto;
- Gerenciamento de dependências superior, incluindo atualização automática, encerramentos de dependência (também conhecidos como dependências transitivas);
- Capaz de trabalhar facilmente com vários projetos ao mesmo tempo;
- Um grande e crescente repositório de bibliotecas e metadados para usar fora da caixa e arranjos com os maiores projetos de código aberto para disponibilidade em tempo real de seus últimos lançamentos.

MAVEN

- Extensível, com a capacidade de escrever facilmente plugins em Java ou linguagens de script;
- Acesso instantâneo a novos recursos com pouca ou nenhuma configuração extra;
- Gerenciamento de releases e publicação de distribuição: Sem muita configuração adicional, o Maven se integrará ao seu sistema de controle de código-fonte e gerenciará o lançamento de um projeto com base em uma determinada tag;
- Gerenciamento de dependências: o Maven incentiva o uso de um repositório central de JARs e outras dependências. O Maven vem com um mecanismo que os clientes do seu projeto podem usar para baixar quaisquer JARs necessários para construir seu projeto a partir de um repositório JAR central.

Repositórios

- **Local**

Dependências estão na sua máquina, dentro da pasta ***\$HOME/.m2/repository***

- Interno

- Externo

Repositórios

- Local

- **Interno**

- Externo

Dependências estão em um servidor interno da empresa. Em uma espécie de intranet, ou seja, somente computadores da empresa tem acesso a ele e podem baixar dependências contidas nele. Com isso, pode-se desenvolver projetos, libs que só serão enxergadas dentro da empresa.

Repositórios

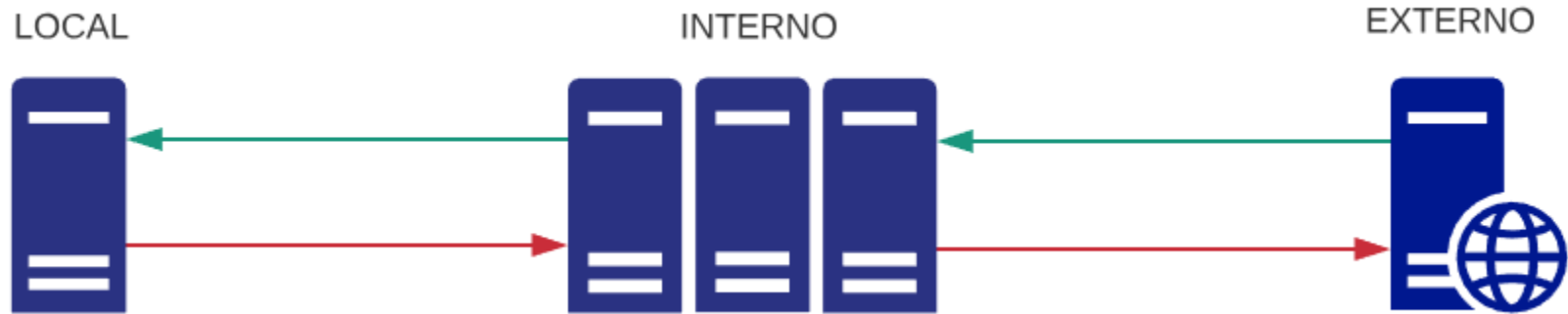
- Local

- Interno

- **Externo**

Dependências estão em servidores na internet, onde podem ser acessadas por qualquer máquina na rede mundial.

Repositórios



Ciclo de vida

- Default
- Clean
- Site

Ciclo de vida - Default

- **initialize**: prepara o build, configurando propriedades, criando diretórios, etc.
- **process-resources**: processa e copia os resources para o diretório das classes compiladas.
- **compile**: compila os códigos-fontes de produção.
- **test-compile**: compila os códigos-fonte de teste.
- **test**: executa os testes e cria os relatórios.
- **package**: empacota os artefatos do build em um arquivo distribuível, como um JAR, por exemplo.
- **install**: salva no repositório local(/home/username/.m2/repository) o arquivo distribuível gerado na fase package.

Ciclo de vida - clean

- **pre-clean:** executa ações necessárias antes da limpeza do build anterior.
- **clean:** remove os artefatos gerados no build anterior.

Ciclo de vida - site

- **site:** cria as páginas web e a documentação do projeto.
- **site-deploy:** faz o deploy(implantação) da documentação e das páginas web geradas na fase site em um servidor.

Ciclo de vida - default: JAR e WAR

- **process-resources**
- **compile**
- **process-test-resources**
- **test-compile**
- **test**
- **package**
- **install**
- **deploy**

POM

Comandos

`mvn clean -DskipTests`

`mvn clean install`

`mvn clean package`



SPRING



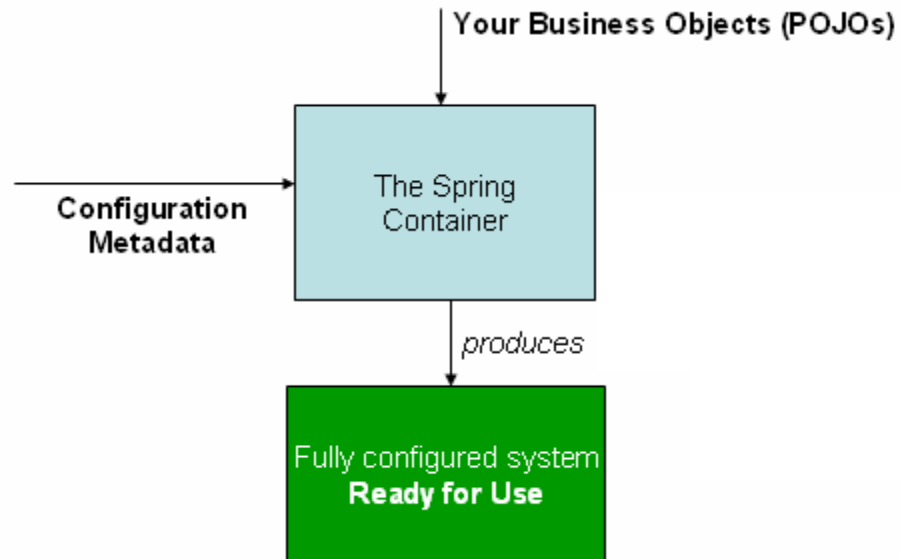
Considerações Iniciais

- Framework
- Spring boot e convenção sobre configuração
- Container IOC
- Beans

Injeção de dependências

- Um processo pelo qual os objetos definem suas dependências (ou seja, os outros objetos com os quais trabalham) somente através de argumentos de construtor, argumentos para um método de fábrica ou propriedades que são configuradas na instância de objeto após serem construídas ou retornadas de um método de fábrica.
- O contêiner então injeta essas dependências quando cria o bean

Injeção de dependências



Beans

- Componentes reutilizáveis
- Encapsulados
- Serializáveis

Anotações

- Metadados;
- Evita o uso excessivo de XMLs de configuração;
- Utilizadas em classes, atributos e métodos;
- Não fazem parte da classe, apenas descreve algo sobre ela;
- Em geral, portanto, usamos anotações para criar configurações nos nossos artefatos com objetivo de que depois essas anotações sejam lidas e processadas por alguém interessado naquelas informações.

Desmistificando algumas anotações...

@Configuration: indica que determinada classe possui métodos que expõe novos beans;

@Controller: Associada com classes que possuem métodos que processam requests numa aplicação web;

@Repository: Associada com classes que isolam o acesso aos dados da sua aplicação. Comumente associada a DAO's;

@Service: Associada com classes que representam a ideia do Service do Domain Driven Design;

Desmistificando algumas anotações...

@Component: indica que uma classe vai ser gerenciada pelo container do Spring;

@ComponentScan: Em geral você a usa em classes de configuração(@Configuration) indicando quais pacotes ou classes devem ser escaneadas pelo Spring para que essa configuração funcione;

@Bean: Anotação utilizada em cima dos métodos de uma classe, geralmente marcada com @Configuration, indicando que o Spring deve invocar aquele método e gerenciar o objeto retornado por ele;

Desmistificando algumas anotações...

@Autowired: Anotação utilizada para marcar o ponto de injeção na sua classe. Você pode colocá-la sobre atributos ou sobre o seu construtor com argumentos;

@Scope: Annotation utilizada para marcar o tempo de vida de um objeto gerenciado pelo container. Pode ser utilizada em classes anotadas com @Component, ou alguma de suas derivações. Além disso também pode usada em métodos anotados com @Bean;

@Primary: Utilizada para informar qual bean padrão retornar, caso você tenha dois métodos anotados com **@Bean** e com ambos retornando o mesmo tipo de objeto;

Desmistificando algumas anotações...

@Profile: Indica em qual profile tal bean deve ser carregado. Muito comum quando você tem classes que só devem ser carregadas em ambiente de dev ou de produção;

@SpringBootApplication: Ponto de entrada do spring boot para tentar carregar todas as outras configurações dentro do projeto.

CONFIGURAÇÕES SPRING

Arquivos XML e Classes

Arquivos XML

Anotação @Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="singleTask" class="com.example.springtask.SingleTask">
    <constructor-arg value="12345678" type="java.lang.Integer"></constructor-arg>
    <constructor-arg value="task description"></constructor-arg>
  </bean>

</beans>
```

O XML acima pode ser importado utilizando-se a anotação
`@ImportResource`

```
@AutoWired
SingleTask singleTask;
```

Arquivos XML e Classes

Arquivos XML

Anotação @Configuration

```
@Import(TaskConfiguration2.class)
@Configuration
public class TaskConfiguration {

    @Bean
    public SingleTask singleTask(){
        return new SingleTask("12345678","task description");
    }
}

@Configuration
public class TaskConfiguration2 {
    //methods and configs
}
```

Autoconfiguração

Spring autoconfigura a aplicação, baseando-se nas dependências declaradas no POM;

@SpringBootApplication

@EnableAutoConfiguration

Autoconfiguração

Spring autoconfigura a aplicação, baseando-se nas dependências declaradas no POM;

A autoconfiguração não é invasiva, ou seja, você pode defini-las manualmente e gradativamente.

Por exemplo, caso adicionemos nosso próprio bean datasource, o suporte padrão do banco de dados incorporado será removido.

```
spring.datasource.url=jdbc:mysql://mysql:3306/mysql  
spring.datasource.username=root  
spring.datasource.password=yourpassword  
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
```

Autoconfiguração

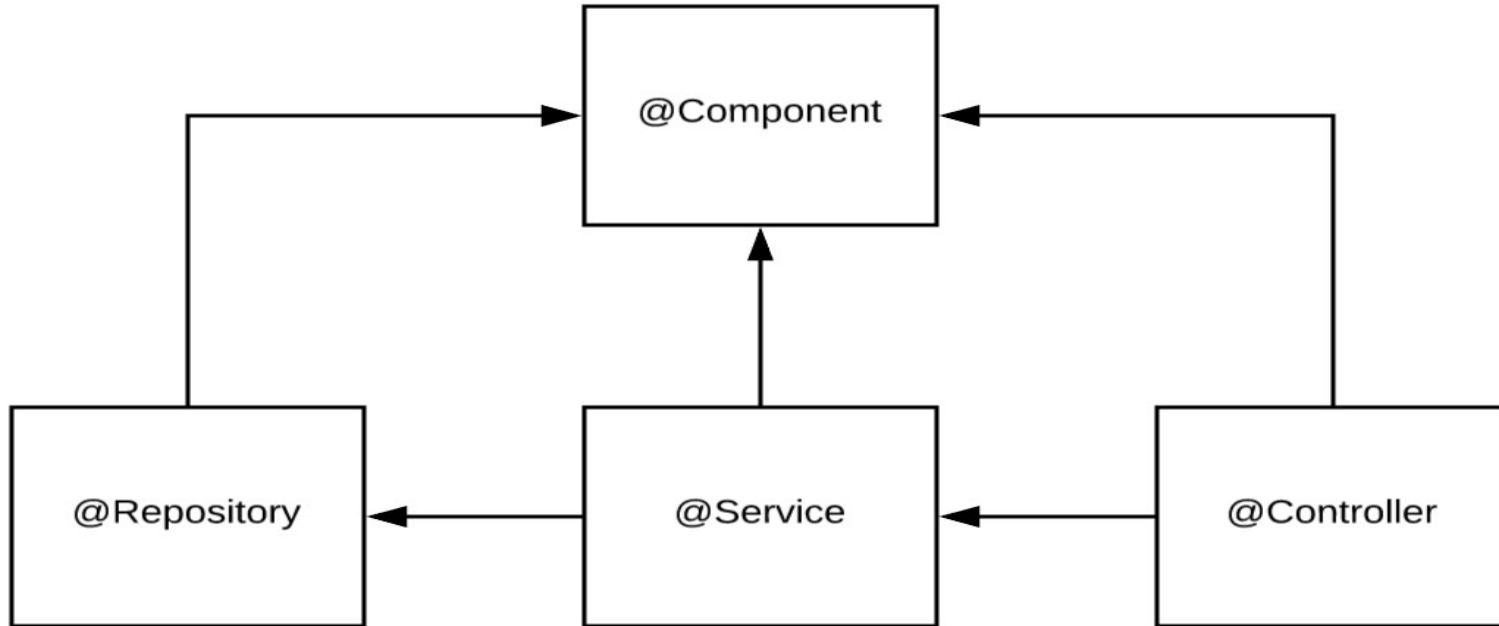
Spring autoconfigura a aplicação, baseando-se nas dependências declaradas no POM;

A autoconfiguração não é invasiva, ou seja, você pode defini-las manualmente e gradativamente;

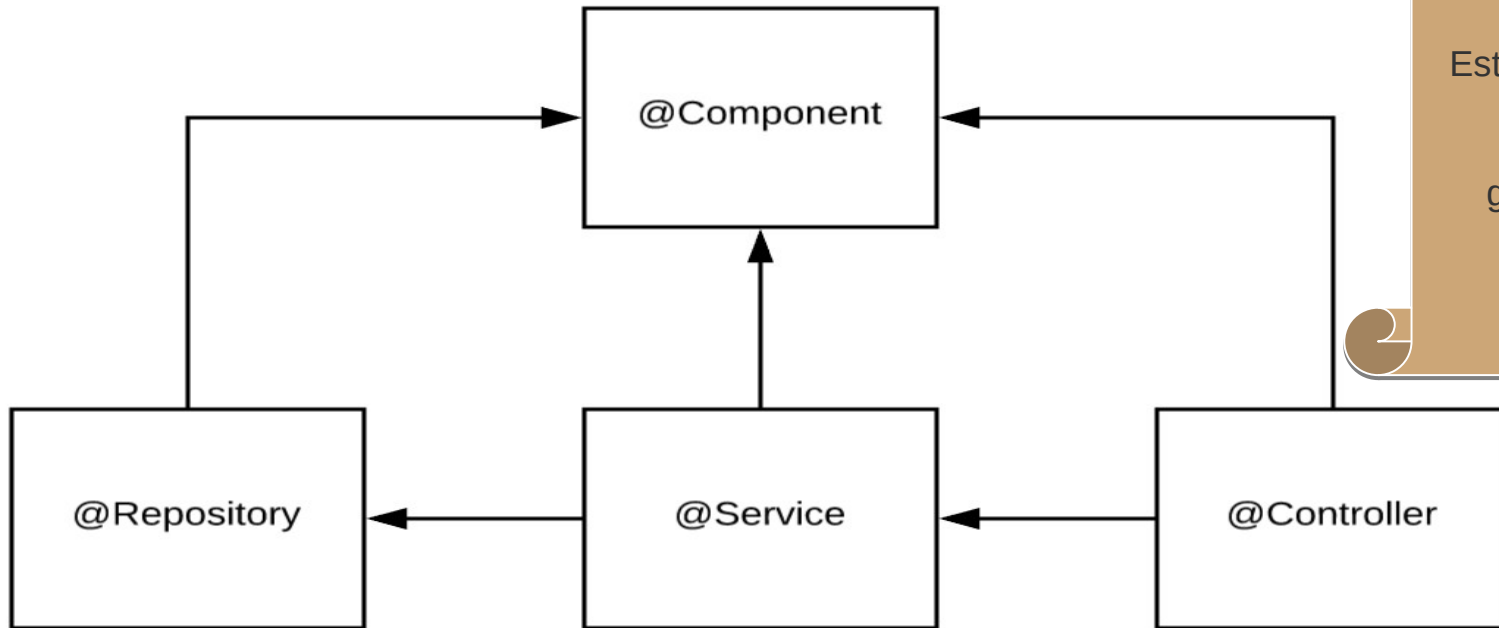
Classes de autoconfiguração podem ser desabilitadas.

```
@Configuration
@EnableAutoConfiguration(exclude =
{DataSourceAutoConfiguration.class})
public class TaskConfiguration2 {
    //methods and configs
}
```

Generalização e especialização de componentes



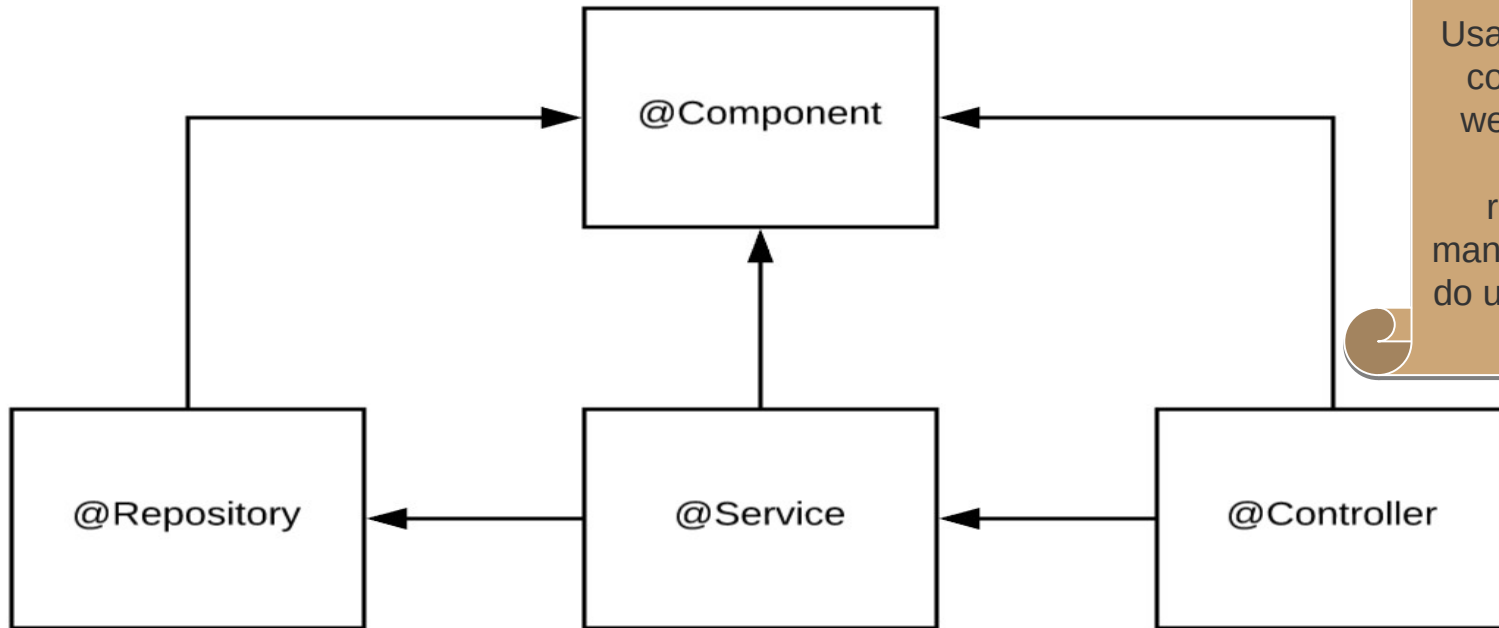
Generalização e especialização de componentes



@Component

Estereótipo genérico
para qualquer
componente
gerenciado pelo
Spring

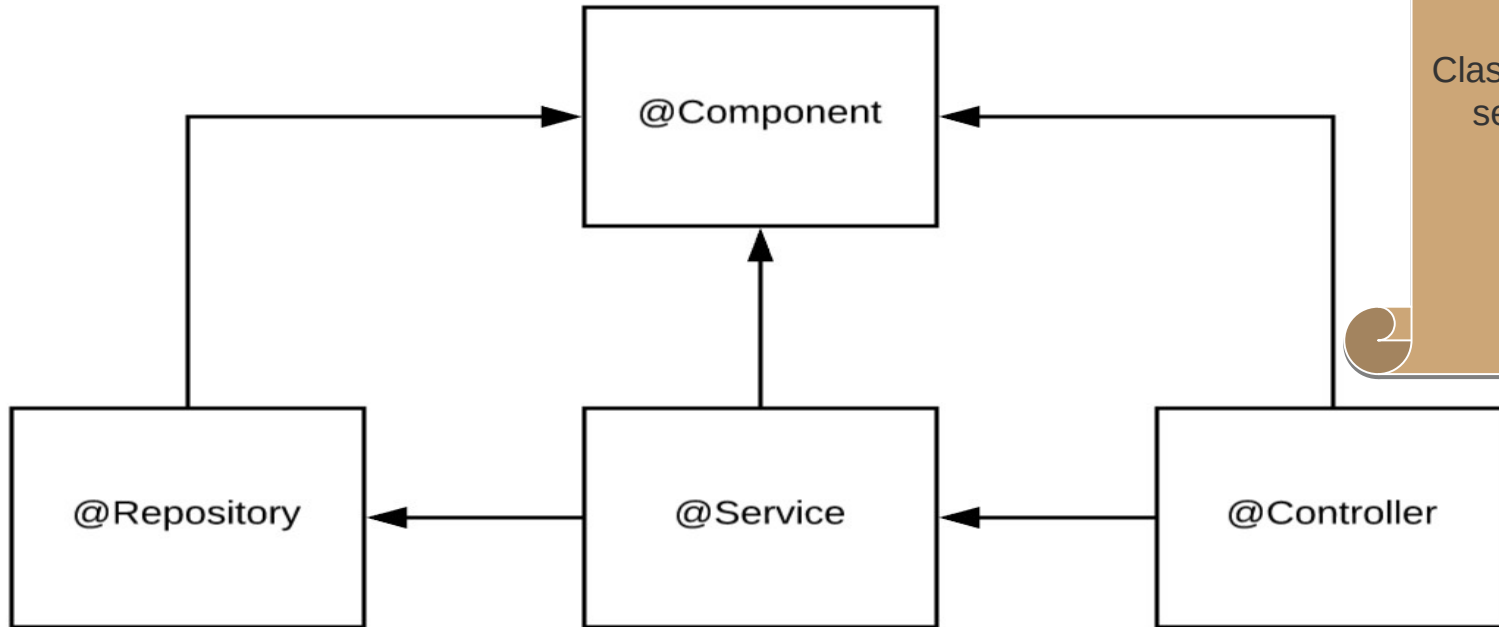
Generalização e especialização de componentes



@Controller

Usado principalmente com aplicativos da web ou serviços da web REST responsável por manipular a solicitação do usuário e retornar a resposta.

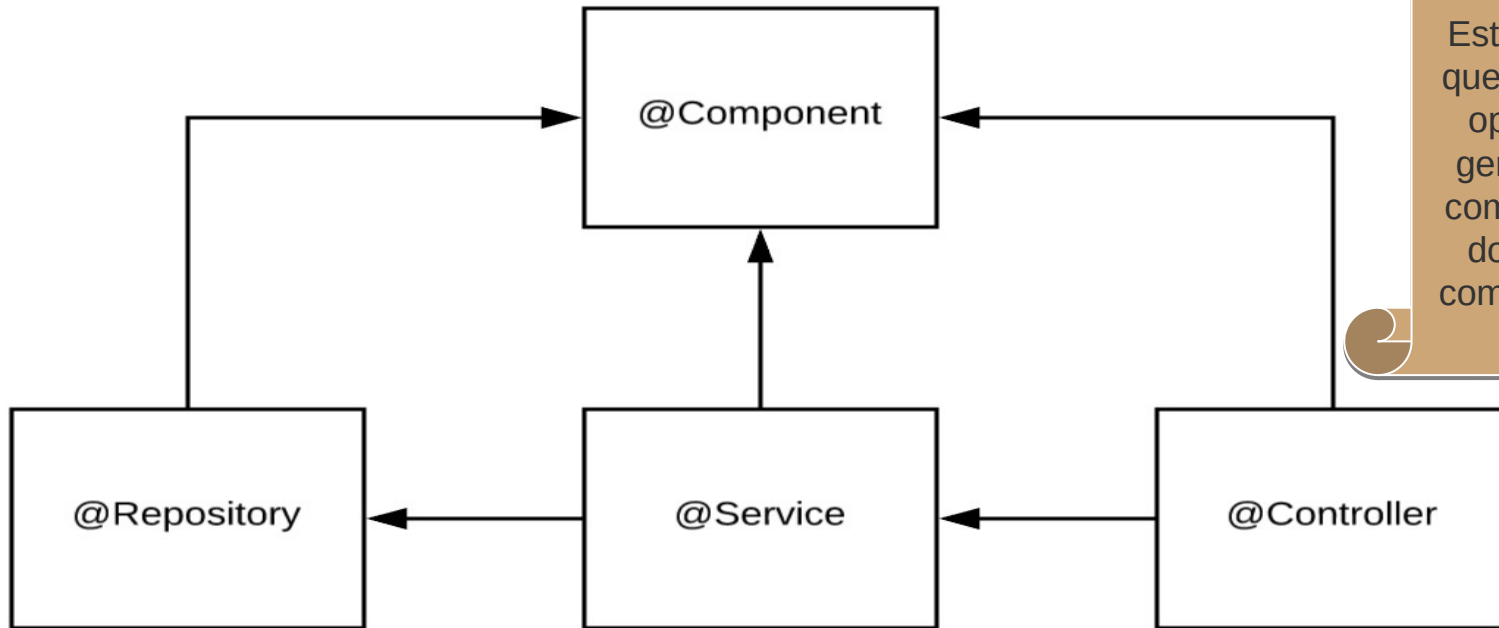
Generalização e especialização de componentes



@Service

Classes na camada de
serviço (negócio).

Generalização e especialização de componentes



@Repository

Esta anotação indica que a classe lida com operações CRUD, geralmente é usada com implementações do DAO que lidam com tabelas de banco de dados.

CAMADA DE NEGÓCIO

Services

Implementação das regras de negócio específicas;

Costuma-se fazer utilização de classes de gerenciamento de dados, outros services ou outras classes utilitárias;

@Service

Cuidados na modelagem das camadas

- Proteja os pacotes “services” e “model” do excesso de dependências;
- As outras camadas utilizaram as classes definidas nesses pacotes e não o contrário;
- Evite a dependência de frameworks nessas camadas.

```
package br.com.ithappens.engenharia.task.services;

import java.io.IOException;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import br.com.ithappens.engenharia.task.mapper.EnvioMapper;
import br.com.ithappens.engenharia.task.mapper.LogEnvioMapper;
import br.com.ithappens.engenharia.task.mapper.SMSMapper;
import br.com.ithappens.engenharia.task.model.Envio;
import br.com.ithappens.engenharia.task.model.LimiteEnvio;
import br.com.ithappens.engenharia.task.model.LogEnvio;
import br.com.ithappens.engenharia.task.model.SMS;
import br.com.ithappens.engenharia.task.model.Sistema;
import br.com.ithappens.engenharia.task.model.exception.EnvioException;
import br.com.ithappens.engenharia.task.model.exception.ExceptionMessage;
import br.com.ithappens.engenharia.task.model.exception.ServiceException;
import br.com.ithappens.engenharia.task.utils.EnvioSMSUtil;
import br.com.ithappens.engenharia.task.utils.TelegramUtils;
import lombok.AllArgsConstructor;
import lombok.extern.log4j.Log4j2;

@Service
@AllArgsConstructor
@Log4j2
public class EnvioSMSService {
```

PERSISTÊNCIA DE DADOS

Configuração de datasource

A partir do endereço `spring.datasource.*` pode ser configurado o datasource externo.

Exemplo:

```
spring.datasource.url=jdbc:mysql://localhost/test
```

```
spring.datasource.username=dbuser
```

```
spring.datasource.password=dbpass
```

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```


MyBatis

- É uma estrutura de persistência com suporte a SQL customizado, procedimentos armazenados e mapeamentos avançados;
- Elimina quase todo o código JDBC e configuração manual de parâmetros e recuperação de resultados;
- Pode usar XML simples ou Anotações para primitivas de configuração e mapa, interfaces de Mapa e Java POJOs para registros de banco de dados.

MyBatis Spring

Permite que o MyBatis:

- Participe de transações Spring;
- Cuide da construção de mapeadores MyBatis e `SQLSession` e injetá-los em outros beans;
- Traduza exceções MyBatis em `Spring DataAccessException`;
- Construir seu código de aplicação livre de dependências em MyBatis, Spring ou MyBatis-Spring.

MyBatis Spring - Configuração

@Bean

```
public SqlSessionFactoryBean sqlSessionFactory(final DataSource oneDataSource)
    throws Exception
{
    SqlSessionFactoryBean factoryBean = new SqlSessionFactoryBean();
    factoryBean.setDataSource(oneDataSource);
    return factoryBean;
}
```

@Bean

```
public MapperFactoryBean<UserMapper> userMapper(SqlSessionFactoryBean
sqlSessionFactory)
    throws Exception
{
    MapperFactoryBean<UserMapper> factoryBean = new
MapperFactoryBean<>(UserMapper.class);
    factoryBean.setSqlSessionFactory(sqlSessionFactory.getObject());
    return factoryBean;
}
```

MyBatis Spring - Declaração Mapper

@Mapper

@Repository

public interface UserMapper {

@Select("SELECT * FROM users WHERE id = #{userId}")

User getUser(@Param("userId") String userId);

}

MyBatis Spring - Declaração Mapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="br.com.ithappens.mapper.UserMapper">

  <select id="getUser" resultType="br.com.ithappens.model.User">
    SELECT
      1 AS id,
      'description' AS description
  </select>
</mapper>
```

MyBatis Spring - Mapper XML

Elementos que compõem a estrutura geral de um mapper XML

- Cache;
- Cache-ref
- ResultMap;
- Sql;
- Insert;
- Update;
- Delete;
- Select.

MyBatis Spring - Mapper XML

Domínio de exemplo:
classe Empresa.

```
public class Empresa {  
    private Integer id;  
    private String descricao;  
    private Long cnpj;  
    private String razaoSocial;  
}
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<select id="getEmpresas" resultType="br.com.treinamento.Empresa">
```

```
    SELECT
```

```
        id AS id,
```

```
        descricao AS descricao,
```

```
        cnpj AS cnpj,
```

```
        razaoSocial as razaoSocial
```

```
    FROM tbl_empresa
```

```
</select>
```

```
List<Empresa> getEmpresas();
```


MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<select id="buscarEmpresa" resultType="br.com.treinamento.Empresa">
```

```
    SELECT
```

```
        id AS id,
```

```
        descricao AS descricao,
```

```
        cnpj AS cnpj,
```

```
        razaoSocial as razaoSocial
```

```
    FROM tbl_empresa
```

```
    WHERE id=#{empresald}
```

```
</select>
```

```
Empresa buscarEmpresa(@Param("empresald") Long empresald);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<select
  id="buscarCodigoEmpresas"
  parameterType="int"
  resultType="hashmap"
  resultMap="personResultMap"
  flushCache="false"
  useCache="true"
  timeout="10"
  fetchSize="256"
  statementType="PREPARED"
  resultSetType="FORWARD_ONLY">
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<insert id="inserirEmpresa" parameterType="br.com.treinamento.Empresa">
```

```
    INSERT INTO tbl_empresa
```

```
        (id,descricao,cnpj,razaoSocial)
```

```
    VALUES
```

```
        (
```

```
        #{empresa.id},
```

```
        #{empresa.descricao},
```

```
        #{empresa.cnpj},
```

```
        #{empresa.razaoSocial}
```

```
    )
```

```
</insert>
```

```
void inserirEmpresa(@Param("empresa") Empresa empresa);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<insert id="inserirEmpresa" parameterType="br.com.treinamento.Empresa"
        useGeneratedKeys="true" keyColumn="id" keyProperty="empresa.id"
>
    INSERT INTO tbl_empresa
        (descricao,cnpj,razaoSocial)
    VALUES
        (
            #{empresa.descricao},
            #{empresa.cnpj},
            #{empresa.razaoSocial}
        )
</insert>
```

```
void inserirEmpresa(@Param("empresa") Empresa empresa);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<insert id="inserirEmpresas" parameterType="br.com.treinamento.Empresa"
        useGeneratedKeys="true" keyColumn="id" keyProperty="empresa.id"
>
INSERT INTO tbl_empresa (descricao, cnpj, razaoSocial)
VALUES
<foreach collection="empresas" item="emp" open="(" separator="),( " close=")">
#{emp.descricao},
#{emp.cnpj},
#{emp.razaoSocial}
</foreach>
</insert>
```

```
void inserirEmpresas(@Param("empresas") List<Empresa> empresas);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<insert
  id="inserirEmpresa"
  parameterType="br.com.treinamento.Empresa"
  flushCache="true"
  statementType="PREPARED"
  keyProperty="id"
  keyColumn="empresa.id"
  useGeneratedKeys=""
  timeout="20">
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<update id="atualizarEmpresa" parameterType="br.com.treinamento.Empresa">
<
UPDATE tbl_empresa
SET
descricao = #{empresa.descricao},
cnpj = #{empresa.cnpj},
razaoSocial = #{empresa.razaoSocial}
WHERE
    id = #{empresa.id}
</update>
```

```
void atualizarEmpresa(@Param("empresa") Empresa empresa);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<update  
  id="atualizarEmpresa"  
  parameterType="br.com.treinamento.Empresa"  
  flushCache="true"  
  statementType="PREPARED"  
  timeout="20">
```


MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<delete id="deletarEmpresa" parameterType="br.com.treinamento.Empresa">
```

```
DELETE FROM tbl_empresa  
WHERE id = #{empresa.id}
```

```
</delete>
```

```
void deletarEmpresa(@Param("empresa") Empresa empresa);
```

MyBatis Spring - Mapper XML

SELECT

INSERT

UPDATE

DELETE

```
<delete  
  id="deletarEmpresa"  
  parameterType="br.com.treinamento.Empresa"  
  flushCache="true"  
  statementType="PREPARED"  
  timeout="20">
```

MyBatis Spring - Result Maps

- Mapeamentos Complexos;
- Associações de classes;
- Coleções como atributos.

MyBatis Spring - Result Maps

Cenário para a classe empresa:

```
<resultMap id="ambienteTrabalho" type="br.com.treinamento.Empresa">
  <id column="id" property="id"/>
  <result column="descricao" property="descricao"/>
  <result column="cnpj" property="cnpj"/>
  <result column="razaoSocial" property="razaoSocial"/>
</resultMap>
```

MyBatis Spring - Result Maps

Cenário: Uma filial que, pertencente a uma empresa, possui uma lista de produtos;

MyBatis Spring - Result Maps

```
public class Filial {  
    private Integer id;  
    private String descricao;  
    private Long cnpj;  
    private String razaoSocial;  
    private Empresa empresa;  
    private List<Produto> produtos;  
}
```

```
public class Empresa {  
    private Integer id;  
    private String descricao;  
    private Long cnpj;  
    private String razaoSocial;  
}  
  
public class Produto {  
    private Integer id;  
    private String descricao;  
    private Double valor;  
}
```

MyBatis Spring - Result Maps

```
<resultMap id="filialMap" type="br.com.treinamento.Filial">
```

```
  <result column="id" property="id"/>
```

```
  <result column="descricao" property="descricao"/>
```

```
  <result column="cnpj" property="cnpj"/>
```

```
  <result column="razaoSocial" property="razaoSocial"/>
```

```
  <association property="empresa" javaType="br.com.treinamento.Empresa">
```

```
    <result column="e_id" property="id"/>
```

```
    <result column="e_descricao" property="descricao"/>
```

```
    <result column="e_cnpj" property="cnpj"/>
```

```
    <result column="e_razaoSocial" property="razaoSocial"/>
```

```
  </association>
```

```
</resultMap>
```

MyBatis Spring - Result Maps

```
<select id="buscarFilialEmpresa" resultMap="filialMap">
```

```
    SELECT
```

```
F.id AS id, F.descricao AS descricao, F.cnpj AS cnpj, F.razaoSocial AS razaoSocial,
```

```
E.id AS e_id, E.descricao AS e_descricao, E.cnpj AS e_cnpj, E.razaoSocial AS e_razaoSocial
```

```
FROM tbl_filial F
```

```
INNER JOIN tbl_empresa E
```

```
ON F.id_empresa=E.id
```

```
AND F.id=#{filial.id}
```

```
</select>
```

```
void buscarFilialEmpresa(@Param("filial") Filial filial);
```


MyBatis Spring - Result Maps

```
<resultMap id="filialMap" type="br.com.treinamento.Filial">
```

```
  <result column="id" property="id"/>
```

```
  <result column="descricao" property="descricao"/>
```

```
  <result column="cnpj" property="cnpj"/>
```

```
  <result column="razaoSocial" property="razaoSocial"/>
```

```
  <collection property="produtos" javaType="ArrayList"
```

```
    ofType="br.com.treinamento.Produto" select="buscarProdutosFilial"
```

```
    column="id" />
```

```
</resultMap>
```

MyBatis Spring - Result Maps

```
<select id="buscarProdutosFilial" resultMap="produtoMapper">  
  SELECT  
    id as id,  
    descricao as descricao,  
    valor as valor  
  FROM tbl_produto  
  WHERE filial_id=#{filial.id}  
</select>
```

```
void buscarProdutosFilial(@Param("filial") Filial filial);
```

MyBatis Spring - Result Maps

```
<resultMap id="filialMap" type="br.com.treinamento.Filial">
  <result column="id" property="id"/>
  <result column="descricao" property="descricao"/>
  <result column="cnpj" property="cnpj"/>
  <result column="razaoSocial" property="razaoSocial"/>
  <association id="empresa" type="br.com.treinamento.Empresa">
    <result column="e_id" property="id"/>
    <result column="e_descricao" property="descricao"/>
    <result column="e_cnpj" property="cnpj"/>
    <result column="e_razaoSocial" property="razaoSocial"/>
  </association>
  <collection property="produtos" javaType="ArrayList"
    ofType="br.com.treinamento.Produto" select="buscarProdutosFilial" column="id"/>
</resultMap>
```

REST

Apresentação

Descrito por Roy Fielding, um dos principais criadores do protocolo HTTP, em sua tese de doutorado;

Restrições:

- Separação cliente-servidor;
- Cada solicitação do cliente para o servidor deve conter todas as informações necessárias para atender a solicitação;
- Não pode aproveitar qualquer contexto armazenado no servidor;
- Pode Usar Cache.

Arquitetura REST

Data Elements	Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image, JSON, XML
representation metadata	media type, last-modified time
representation metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Métodos HTTP

Método	exemplo	Código de Resposta
GET	HTTP GET http://localhost/empresas HTTP GET http://localhost/empresas/1	200, 400 ou 404
POST	HTTP POST http://localhost/empresas	200, 201 ou 204
PUT	HTTP PUT http://localhost/empresas/1	200, 201 ou 204
DELETE	HTTP DELETE http://localhost/empresas/1	200, 202 ou 204

Métodos HTTP

Conforme exemplo anterior, a URN não varia, pois todas as requisições são feitas para a URL base mais o identificador do recurso (empresas);

Isso se dá pela utilização semântica dos verbos HTTP, em que o backend deve executar ações distintas baseadas no verbo, como por exemplo um POST para empresas é para salvar um recurso, assim como um PUT para atualizar, um GET para buscar e um DELETE para deletar, portanto a URI não precisa de variações em sua base.

Métodos HTTP - Recomendações

Mapear os endpoints baseando-se no recurso, por exemplo, se estiver trabalhando com empresa, utilizar o */empresas*, se estiver trabalhando com produto, utilizar o */produtos*, e assim sucessivamente;

Jamais montar URIs expondo as ações que serão feitas, por exemplo: */empresas/salvar*. No cenário anterior, basta que se envie um POST para */empresas*, que automaticamente se entende que é uma requisição de salvar recurso;

Controllers

São o ponto de entrada dos endpoints que proveremos em nossos serviços;

Por meio deles, nós recebemos a requisição, processamos a regra de negócio e retornamos uma resposta à solicitação;

É essencial que controllers traduzam as exceções ou adversidades em status HTTP e mensagens de resposta para que, independente de qualquer fluxo alternativo ou exceção, sempre seja retornada uma resposta à solicitação;

Controllers - Exemplo POST

@RestController

@RequestMapping("/empresas")

public class EmpresaController {

@Autowired

EmpresaService empresaService;

@PostMapping

public ResponseEntity salvar(@RequestBody Empresa empresa){
 empresaService.salvar(empresa);
 return ResponseEntity.ok("salvo com sucesso!");
}
}

Controllers - Exemplo GET

@RestController

@RequestMapping("/empresas")

public class EmpresaController {

@GetMapping

public ResponseEntity buscar(){

List<Empresa> empresas = empresaService.buscarEmpresas();

return ResponseEntity.ok(empresas);

}

}

Controllers - Exemplo GET

```
@RestController
```

```
@RequestMapping("/empresas")
```

```
public class EmpresaController {
```

```
@GetMapping(value =("/{id}/{cnpj}")
```

```
public ResponseEntity buscar(@PathVariable("id") Long  
id, @PathVariable("cnpj") String cnpj){
```

```
    Empresa empresa = empresaMapper.buscar(id);
```

```
    return ResponseEntity.ok(empresa);
```

```
}
```

```
}
```

Controllers - Exemplo PUT

```
@RestController
```

```
@RequestMapping("/empresas")
```

```
public class EmpresaController {
```

```
@PutMapping(value =("/{id}")
```

```
public ResponseEntity atualizar(@PathVariable("id") Long id, @RequestBody Empresa  
empresa){
```

```
    empresa.setId(id);
```

```
    empresaService.atualizar(empresa);
```

```
    return ResponseEntity.ok(("Atualizado com sucesso!"));
```

```
}
```

```
}
```

Controllers - Exemplo DELETE

```
@RestController
```

```
@DeleteMapping("/empresas")
```

```
public class EmpresaController {
```

```
@DeleteMapping(value =("/{id}")
```

```
public ResponseEntity salvar(@PathVariable("id") Long id){
```

```
    empresaMapper.deletar(id);
```

```
    return ResponseEntity.ok("Deletado com sucesso!");
```

```
}
```

```
}
```

Tratamento de exceções e Respostas HTTP

O Spring tem um padrão de retorno de mensagem caso alguma exceção seja lançada, exemplo:

```
{  
  "timestamp": "2019-06-25T23:43:38.334+0000",  
  "status": 500,  
  "error": "Internal Server Error",  
  "message": "Empresa Não encontrada",  
  "trace": "br.com.treinamento.utils.exceptions.EmpresaNotFoundException:  
Empresa Não encontrada.....+texto gigante do trace de erro",  
  "path": "/empresas/6"  
}
```


Tratamento de exceções e Respostas HTTP

No exemplo anterior, tentou-se buscar a empresa 6, porém o resultado foi não encontrado. Como é uma exceção estourada, o retorno padrão foi o código HTTP 500 (Internal Server Error), porém o erro real foi de NotFoundException, então, intuitivamente, deveria ser retornado um 404 (Not Found). A maneira mais básica de mapear uma exceção a um código HTTP é utilizando a anotação `@ResponseStatus`:

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "Sem Resultado")
```

```
public class EmpresaNotFoundException extends RuntimeException{  
    public EmpresaNotFoundException(String message) {  
        super(message);  
    }  
}
```

Tratamento de exceções e Respostas HTTP

A anotação de `ResponseStatus`, escrita no exemplo anterior, mapeia um `HttpStatus` à classe da exceção, ou seja, toda vez que estoura uma exceção do tipo desta, o código HTTP retornado como resposta é o mapeado na classe, conforme exemplo seguinte. Observa-se que o retorno já veio com o código HTTP correto, mas ainda há informações desnecessárias ao usuário.

```
{  
  "timestamp": "2019-06-26T00:06:53.455+0000",  
  "status": 404,  
  "error": "Not Found",  
  "message": "Sem Resultado",  
  "trace": "br.com.treinamento.utils.exceptions.EmpresaNotFoundException: Empresa Não  
encontrada.....+texto gigante do trace de erro",  
  "path": "/empresas/6"  
}
```

Tratamento de exceções e Respostas HTTP

Para customizar a estrutura da mensagem de retorno, problematizada no exemplo anterior, uma abordagem do Spring são os controllers advices, que funcionam como interceptadores de exceção, onde dentro deles pode-se converter um padrão de mensagem a outro. Um advice é uma classe controller, que possui vários métodos mapeados às exceções:

@RestController

@ControllerAdvice

```
public class EmpresaController extends ResponseEntityExceptionHandler{  
  
}
```

Tratamento de exceções e Respostas HTTP

@RestController

@ControllerAdvice

```
public class EmpresaController extends ResponseEntityExceptionHandler{
    @ExceptionHandler(EmpresaNotFoundException.class)
    public final ResponseEntity<ExceptionResponse> notFound(EmpresaNotFoundException
ex) {
        ExceptionResponse exceptionResponse = new ExceptionResponse(
            ex.getMessage()
        );
        return new ResponseEntity<ExceptionResponse>(exceptionResponse,
            HttpStatus.NOT_ACCEPTABLE);
    }
}
```

Tratamento de exceções e Respostas HTTP

No código anterior tem-se um controller advice estendendo a classe `ResponseEntityExceptionHandler`. Os handlers são interceptadores utilizados pelo spring para gerenciar essa captura;

Dentro, possui-se um método anotado com `@ExceptionHandler`, passando-se a classe da exceção `EmpresaNotFoundException` como parâmetro. Isso indica que, quando estourar uma exceção do tipo dela, esse método vai ser chamado, fará o tratamento necessário e retornará algo como resposta.

Tratando-se de um handler `ResponseEntity`, o método justamente retorna um `ResponseEntity` que contém o objeto customizado, que será transformado em mensagem para retorno da requisição. Esse objeto é da classe `ExceptionResponse`, uma classe criada com parâmetros customizados para retornar, descrita no slide seguinte.

Tratamento de exceções e Respostas HTTP

```
public class ExceptionResponse {  
    private String mensagem;  
  
    public ExceptionResponse(String message) {  
        super();  
        this.mensagem = message;  
    }  
  
    public String getMensagem() {  
        return mensagem;  
    }  
}
```

Tratamento de exceções e Respostas HTTP

Após habilitado o controller advice, e fazendo a mesma requisição GET para `/empresas/6`, o retorno vem customizado, conforme mostrado abaixo:

```
{  
  "mensagem": "Empresa Não Encontrada"  
}
```

Isso justamente porque o Advice interceptou a exceção e a transformou no objeto da classe `ExceptionResponse`, que contem a estrutura acima, quando convertida para JSON.