PÓS-GRADUAÇÃO ALFA

UNIALFA
CENTRO UNIVERSITÁRIO
ALVES FARIA

Diego Américo Guedes

PROGRAMAÇÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

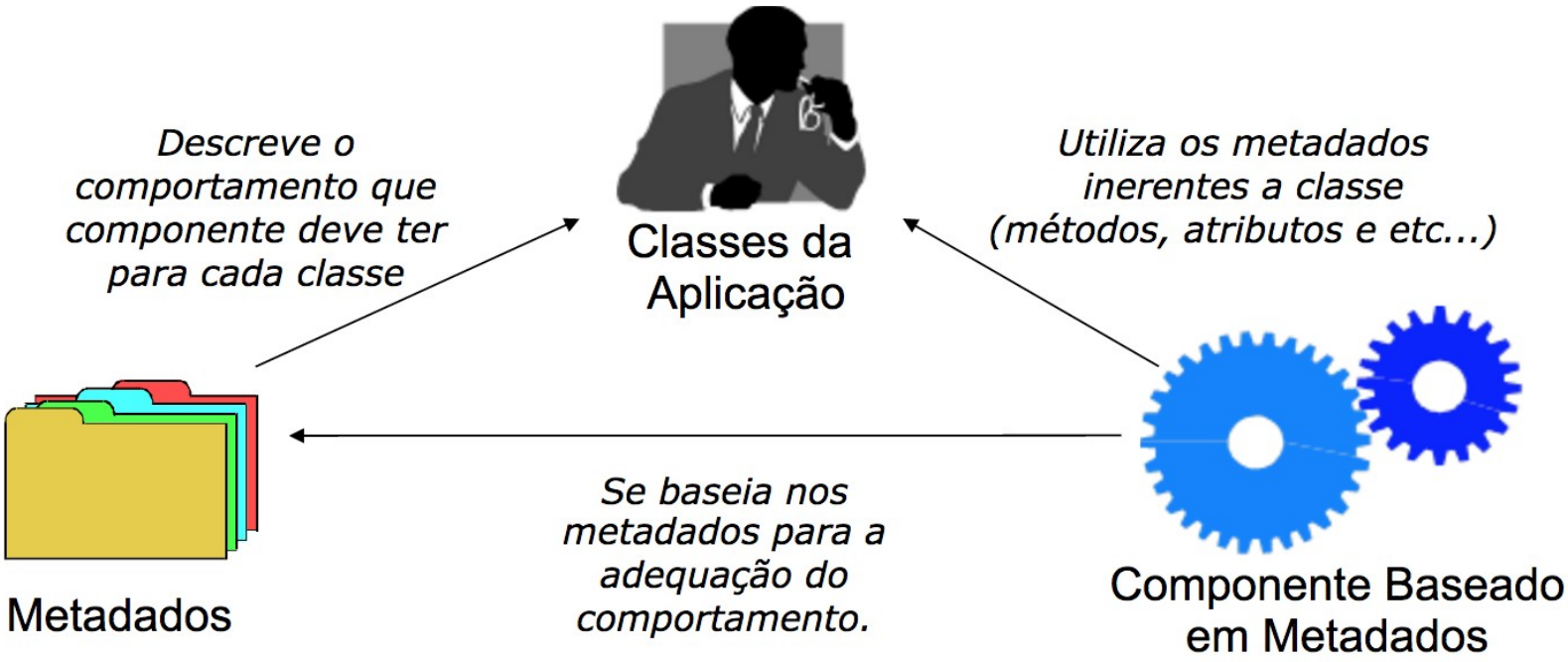
Framework Baseado em Metadados

Prof. Me. Diego Guedes

Definição

- Framework nos quais o seu comportamento é configurado por meio de metadados (dados que definem outros dados)

Uso de metadados



- Os metadados podem ser da aplicação (onde eles configuram características do componentes da aplicação) ou de objetos (onde eles configuram as características de cada objeto da aplicação).

Vantagens

- É possível personalizar o comportamento de um componente ou framework de acordo com as necessidades da aplicação
- Aumento de reuso do componente que utilizar os metadados, devido a possibilidade de ser configurado
- Diminuição de código repetitivo e braçal, o que diminui a possibilidade de defeitos muitas vezes difíceis de serem encontrados
- Aumento da produtividade da equipe
- Aumenta a flexibilidade da aplicação devido ao caráter configurável do componente
- Dependendo da forma que os metadados forem armazenados, eles podem ser alterados sem a necessidade de recompilar o código fonte

Exemplos

- **Hibernate**: uso de metadados para mapear o paradigma orientado a objetos para o paradigma relacional, provendo um componente genérico para o acesso a dados
- **Spring**: um framework baseado em metadados para o gerenciamento da injeção de dependência e inversão de controle em uma aplicação
- **EJB 3**: Uso de metadados para gerenciamento do ciclo de vida, controle de transações, segurança, injeção de dependência e outras coisas.

Generics e Annotation

- Dado um arquivo de configuração, deve-se criar uma classe com o respectiva definição do arquivo
- Assim, é impraticável fazer uma classe prévia para todas as possíveis combinações de palavras da classe

Metaprogramação

- O ideal é utilizar metaprogramação somente em casos onde só se tenha acesso a determinadas informações em tempo de execução.

Metaprogramação. Quando usar?

- Suponha que você tenha um servidor back-end Java que receba requisições da web e retorne um XML correspondente
- Por exemplo, uma requisição da forma “/conta/lista”

```
String path = "/conta/lista";
String[] subPaths = path.replaceFirst("/", "")
                        .split("/");

if (subPaths[0].equals("conta")) { // conta
    ContaController conta = new ContaController();

    if (subPaths[1].equals("filtra")) {
        conta.filtra();
    } else if (subPaths[1].equals("lista")) {
        conta.lista();
    } //outro else-if
} else if (subPaths[0].equals("cliente")) { //
/cliente
    // ifs aninhados para descobrir o método
} // outros else-if
```

Metaprogramação. Quando usar?

- Para gerar o XML

```
String xml = "";
if (objeto instanceof Conta) {
    Conta conta = (Conta)objeto;
     Integer numero = conta.getNumero();
     String titular = conta.getTitutal();
     Double saldo = conta.getSaldo();

     xml = "<conta>" +
             "<numero>" + numero + "</numero>" +
             "<titular>" + titular + "</titular>"
+
             "<saldo>" + saldo + "</saldo>" +
         "</conta>";
}else if (objeto instanceof Cliente) {
    // lógica para gerar o XML
} //outros else-if
```

Metaprogramação. Quando usar?

- Como nossa entrada é realizada de forma dinâmica, temos esses dois problemas
  - No primeiro caso, podemos ter diversos tipos de URL sendo passadas
  - No segundo caso, podemos ter diversos tipos de objetos que deverão ser criados

- Como resolver esse problema?

Metaprogramação. Quando usar?

- Como nossa entrada é realizada de forma dinâmica, temos esses dois problemas
  - No primeiro caso, podemos ter diversos tipos de URL sendo passadas
  - No segundo caso, podemos ter diversos tipo de objetos que deverão ser criados

- Como resolver esse problema?
  - Em Java, usando a API Reflection ;-)

Metaprogramação. Quando usar?

- Para se instanciar um objeto de acordo com uma entrada dinâmica, usa-se a classe Class<T> em Java
- Há 3 formas de obtê-la
  1) .class
  2) getClass()
  3) forName("CAMINHO")

```
ContaCorrente contaCorrente = new ContaCorrente();

Class<Conta> conta1 = Conta.class;
Class<? extends Conta> conta2 = contaCorrente.getClass();
Class<?> conta3 = Class.forName("br.com.diego.banco.controle.Conta");
```

Metaprogramação. Quando usar?

- Agora, vamos criar um objeto da classe com o método `newInstance()`

```
    Conta conta1Instanciada = conta1.newInstance();
    Object conta3Instanciada = conta3.newInstance();

    System.out.println(conta1Instanciada instanceof
Conta);
    System.out.println(conta3Instanciada instanceof
Conta);
```

Classe Constructor<T>

- Note que o método `newInstance()` está depreciado
- Para evitar isso, devemos utilizar a classe Construtor<T>
- Existem 4 formas de se obter um objeto da classe Constructor<T>, e todas elas são por meio da classe Class<T>
  1) getConstructor(s)
  2) getConstrutor(Class<T> args)
  3) getDeclaredConstructors()
  4) getDeclaredConstructor(Class<T> args)
- Uma vez obtido, é só chamar newInstance()
  - Se for chamar newInstance() para construtores privados, deve-se setar como true para manipular o construtor - setAcessible(true)

Classe Constructor<T>

```
Class<ContaCorrente> contaCorrente1 = ContaCorrente.class;
Constructor<ContaCorrente> construtorContaCorrente1 = contaCorrente1.getConstructor(String.class);

ContaCorrente oContaCorrente = construtorContaCorrente1.newInstance("Diego");
System.out.println(oContaCorrente);
```

Classe Method

- Existem 4 formas de se obter um objeto da classe Method, e todas elas são por meio da classe Class<T>
  1) getMethods()
  2) getMethod(String nome, Class<?> ...tiposArgs)
  3) getDeclaredMethods()
  4) getDeclaredMethod(String nome, Class<?>...tipoArgs)
- Uma vez obtido, é só chamar invoke(Object obj, Object... args)
  - Se for chamar invoke() para métodos privados, deve-se setar como true para poder utilizar o método - setAcessible(true)

Classe Method

```
Class<ContaCorrente> contaCorrente1 = ContaCorrente.class;
Constructor<ContaCorrente> construtorContaCorrente1 = contaCorrente1.getConstructor(Double.class);
ContaCorrente oContaCorrente = construtorContaCorrente1.newInstance(2.3);

for (Method m : contaCorrente1.getMethods()) {
System.out.println(m);
}
System.out.println("");
for (Method m : contaCorrente1.getDeclaredMethods()) {
System.out.println(m);
}

Method getTaxa = contaCorrente1.getDeclaredMethod("getTaxa");

System.out.println("");
System.out.println(getTaxa.invoke(oContaCorrente));
```

Classe Field

- Existem 4 formas de se obter um objeto da classe Field (Campo/Atributo) de uma classe, e todas elas são por meio da classe Class<T>
  1) getFields()
  2) getField(String nome)
  3) getDeclaredFields()
  4) getDeclaredField(String nome)
- Uma vez obtido:
  - getName() para pegar o nome do atributo
  - get(OBJETO) para pegar o valor do atributo
    - Se for privado, deve-se setar como true para poder utilizar o valor do atributo - setAcessible(true)

Classe Field

```
ContaCorrente cc = new ContaCorrente(2.1);
Class<? extends ContaCorrente> classeCC = cc.getClass();

for (Field atributo : classeCC.getFields()) {
atributo.setAccessible(true);
System.out.println(atributo.getName() + " - " + atributo.get(cc));
}
System.out.println("");

Conta conta = new Conta("diego", 2.3);
Class<? extends Conta> classeConta = conta.getClass();
for (Field atributo : classeConta.getDeclaredFields()) {
atributo.setAccessible(true);
System.out.println(atributo.getName() + " - " + atributo.get(cc));
}
System.out.println("");
System.out.println(classeConta.getDeclaredField("saldo").getName() + " - "
+ classeConta.getDeclaredField("saldo").get(conta));
```

Metadados em Java – Anotações

- Metadados são dados relativos a uma informação já existente, que é o nosso código, ou seja, dentro de um código Java nada é uma informação relativa a um trecho de código escrito anteriormente.
- Antes da versão 5 do Java, era comum que isso fosse feito por meio de XML
- Agora é feito por annotations
- Vamos criar uma anotação Java chamada NomeTagXml em br.com.diego.conta.playground.anotacao
- Vamos usar a anotação `@Retention` para que nossa anotação seja levada em conta em tempo de execução

```
@Retention(RetentionPolicy.RUNTIME)
```

- Iremos utilizar a anotação @Target para informar onde queremos usar nossa anotação. Usaremos na classe e nos atributos

```
@Target({ElementType.FIELD,ElementType.TYPE})
```

Metadados em Java – Anotações

- Agora, vamos incluir nossa anotação na Classe Conta

```
@NomeTagXml
public class Conta {
```

- Conforme definido no @Target, também podemos utilizar para os atributos

```
@NomeTagXml
private String nome;
```

- Por exemplo, queremos mudar o nome que deverá ser utilizado pela classe

```
@NomeTagXml ("conta")
public class Conta {
```

- Nesse caso, haverá um erro dizendo que o atributo value não está definido.
- Daí, basta ir na anotação e criar o método

```
public String value();
```

- Caso queira criar um nome especializado, basta criar um método na anotação com o nome correspondente

```
@NomeTagXml(nome="titular")
private String nome;
```

  - Na anotação:

```
public String nome();
```

  - De posse do Class<?>, é possível obter as anotações e verificar se alguma anotação específica pertence a classe e pegar o respectivo valor da anotação (no nosso caso, nome)

```
Conta conta = new Conta();
Class<?> classeConta = conta.getClass();
System.out.println(classeConta.getDeclaredAnnotation(NomeTagXml.class).nome());
```

Criando nosso próprio Framework

- Criaremos um framework chamado Summer ;-)

Classe que simula um navegador

```
package br.com.diego.banco;

import java.util.Scanner;
import br.com.diego.summer.Summer;

public class Main {
/**
 * Simula o navegador.
 *
 */
public static void main(String[] args) throws Exception {
Summer summer = null;

try (Scanner s = new Scanner(System.in)) {
String url = s.nextLine();

while (!url.equals("exit")) {
summer = new Summer(url);
Object response = summer.executa();
System.out.println("Resposta: " + response);

url = s.nextLine();
}
}
}
}
```

Classe do Framework – Criando um objeto dinamicamente

```
package br.com.diego.summer;

import java.lang.reflect.InvocationTargetException;

public class Summer {
private String caminhoClasse;

public Summer(String url) {
String[] partesUrl = url.replaceFirst("/", "").split("/");
if (partesUrl[0].compareTo("conta") == 0)
this.caminhoClasse = "br.com.diego.banco.controle.ContaController";
}

public Object executa() {
try {
Class<?> classeControle = Class.forName(caminhoClasse);

Object instanciaControle = classeControle.getConstructor().newInstance();

System.out.println(instanciaControle);
} catch (ClassNotFoundException | InstantiationException | IllegalAccessException
| IllegalArgumentException | NoSuchMethodException | SecurityException e) {
e.printStackTrace();
} catch (InvocationTargetException e) {
e.printStackTrace();
throw new RuntimeException("Erro no construtor", e.getTargetException());
}
return null;
}
}
```

Classe do Framework – Chamando um método dinamicamente

- Vamos chamar dinamicamente o método que foi passado na URL
- Lembrando que a url tem o padrão “/classe/metodo”

```
Object retorno = classeControle.getDeclaredMethod(nomeMetodo).invoke(instanciaControle);
```

Classe Summer – Chamando os métodos com parâmetros

- Vamos trabalhar agora com a chamada de método utilizando os parâmetros da requisição
- Por exemplo, dada a requisição
  - */conta/filtra?nome=diego&cpf=12345678900*
- Queremos tratá-la e chamar o método correspondente em `ContaController`
- Note que o construtor de Summer começou a ficar muito complexo.
- Para isso, iremos criar uma classe específica chamada Request em br.com.diego.summer.protocolo
- Além disso, uma classe QueryParamsBuilder em br.com.diego.summer.protocolo para montar os parâmetros chave/valor

Classe Summer – Chamando os métodos com parâmetros

- Para isso, precisamos
  - 1) Pegar todos os métodos da classe.
  - 2) Filtrar todos os métodos de modo que:
    - 2.1) Tenham o mesmo nome informado pelo usuário;
    - 2.2) Tenham a mesma quantidade de parâmetros passados na URL;
    - 2.3) E que cada um dos parâmetros tenham os mesmos nomes e tipos iguais aos passados na URL.
  - 3) Lançar uma RuntimeException caso nenhum método seja encontrado.

Classe Summer – Chamando os métodos com parâmetros

- 1) Pegar todos os métodos da classe.

```
Stream<Method> metodos = Stream.of(instanciaControle.getClass().getDeclaredMethods());
```

- 2) Filtrar todos os métodos de modo que:
  - 2.1) Tenham o mesmo nome informado pelo usuário;

```
metodos.filter(metodo -> metodo.getName().compareTo(this.request.getMetodo()) == 0
```

  - 2.2) Tenham a mesma quantidade de parâmetros passados na URL;

```
&& metodo.getParameterCount() == this.request.getQueryParams().size()
```

  - 2.3) E que cada um dos parâmetros tenham os mesmos nomes e tipos iguais aos passados na URL.

Classe Summer – Chamando os métodos com parâmetros

- 2) Filtrar todos os métodos de modo que:
  - 2.3) E que cada um dos parâmetros tenham os mesmos nomes e tipos iguais aos passados na URL.

```
&&Stream.of(metodo.getParameters()).allMatch(
param->this.request.getQueryParams().keySet().contains(param.getName())
&& this.request.getQueryParams().get(param.getName()).getClass().equals(param.getType()))
```

- Uma vez que tenha dado certo, vamos pegar a primeira e única ocorrência

```
.findFirst()
```

- 3) Lançar uma RuntimeException caso nenhum método seja encontrado.

```
.orElseThrow(() -> new RuntimeException("Método não encontrado!"));
```

Classe Summer – Chamando os métodos com parâmetros

- Executando a a classe Main e passando `/conta/filtra?nome=diego` teremos novamente a exceção
- O Java está chamando os parâmetros do nosso método filtra() da classe ProdutoController de arg0 e arg1.
- Isso porque, quando o Java gera o bytecode, ele não leva o nome que declaramos na nossa classe - ele faz otimizações e atribui um nome genérico aos nossos parâmetros.
- Para resolvermos esse tipo de problema, precisaríamos utilizar alguma biblioteca que nos possibilitasse manipular nossos parâmetros com seus nomes originais.
- Uma biblioteca bem famosa, bastante utilizada por quem trabalha com Reflection, é a chamada Paranamer.
- No entanto, o Java permite, a partir da versão 8, que recuperemos o nome dos nossos parâmetros usando a própria API de Reflection.

Classe Summer – Chamando os métodos com parâmetros

- Para isso, clicaremos com o botão direito no projeto banco-api e em seguida em "Propriedades".
- Na seção "Java Compiler", selecionaremos a opção "Store information about method parameters (usable via reflection)"

Classe Summer – Chamando os métodos com parâmetros

- Uma vez encontrado o método, devemos fazer a invocação com os parâmetros enviados
- Lembrando que, mesmo que o parâmetros sejam enviados em ordem inversa
  - /conta/filtra?cpf=12345678900&nome=diego
- Devemos invocar a função na ordem correta filtra(nome,cpf)

Classe Summer – Chamando os métodos com parâmetros

- Vamos criar um ArrayList de parâmetros

```
List<Object> parametros = new ArrayList<Object>();
```

- Depois, vamos pegar os parâmetros do método selecionado e ir colocando parâmetro a parâmetro no ArrayList

```
Stream.of(metodoSelecionado.getParameters())
.forEach(p -> parametros.add(this.request.getQueryParams().get(p.getName())));
```

- Por fim, vamos retornar a invocação do método, transformar o ArrayList em um vetor de Objetos

```
return metodoSelecionado.invoke(instanciaControle, parametros.toArray());
```

Classe Summer – Retornando um XML

- Vamos criar uma classe chamada ConversorXML em br.com.diego.summer.xdr que terá um método chamado “convert” que receberá um Object que será convertido em um XML
- Precisamos ver se é um Collection, pois no caso de uma lista, precisaríamos retornar um XML com várias tags <conta>, e um XML válido tem apenas uma tag raiz.

```
package br.com.diego.summer.xdr;

import java.util.Collection;
public class ConversorXML {
public String convert(Object objeto) {
Class<?> classeObjeto = objeto.getClass();
StringBuilder xml = new StringBuilder();
if (objeto instanceof Collection) {
Collection<?> colecao = (Collection<?>) objeto;
xml.append("<lista>");
for (Object o : colecao) {
      xmlRetorno = convert(o);
     xml.append(xmlRetorno);
}
xml.append("</lista>");
}
return xml.toString();
}
}
```

Classe Summer – Retornando um XML

- Caso não seja uma coleção (else), vamos tratar a seguir
- Iniciando o xml com o nome da classe

```
else {
nomeClasse = classeObjeto.getName();
xml.append("<"+nomeClasse+">");
xml.append("</"+nomeClasse+">");
}
```

- O próximo passo e pegar o nome e valor dos atributos

```
for (Field campo : classeObjeto.getDeclaredFields()) {
campo.setAccessible(true);
nomeAtributo = campo.getName();
valorAtributo = campo.get(objeto);
xml.append("<" + nomeAtributo + ">");
xml.append(valorAtributo);
xml.append("</" + nomeAtributo + ">");
}
```

Framework Summer – Trabalhando com anotações

- Vamos agora ao invés de retornar no XML <br.com.diego.banco.modelo.Conta> queremos retornar <conta>
- Para isso, vamos anotar na nossa classe Conta que seu nome é conta.
- Além disso, vamos anotar o nome do atributo nome como “titular”
- Assim, vamos usar a anotação já criar NomeTagXml e colocá-la no pacote br.com.diego.banco.anotacao
- Funcionou?

Framework Summer – Trabalhando com anotações

- Vamos agora ao invés de retornar no XML <br.com.diego.banco.modelo.Conta> queremos retornar <conta>
- Para isso, vamos anotar na nossa classe Conta que seu nome é conta.
- Além disso, vamos anotar o nome do atributo nome como “titular”
- Assim, vamos usar a anotação já criar NomeTagXml e colocá-la no pacote br.com.diego.banco.anotacao do projeto Summer
- Funcionou?
  - Não! Por quê?

Framework Summer – Trabalhando com anotações

- Vamos agora ao invés de retornar no XML <br.com.diego.banco.modelo.Conta> queremos retornar <conta>
- Para isso, vamos anotar na nossa classe Conta que seu nome é conta.
- Além disso, vamos anotar o nome do atributo nome como “titular”
- Assim, vamos usar a anotação já criar NomeTagXml e colocá-la no pacote br.com.diego.banco.anotacao
- Funcionou?
  - Não! Por quê?
    - Porque não avisamos ao Java para usá-lo
- Assim, vamos na classe ConversorXML no método convert para fazer o uso das anotações

Framework Summer – Trabalhando com anotações

- Vamos mudar a atribuição do nomeClasse

```
anotacaoClasse = classeObjeto.getDeclaredAnnotation(NomeTagXml.class);
nomeClasse = anotacaoClasse == null ? classeObjeto.getName() : anotacaoClasse.nome();
```

- Da mesma forma, vamos mudar a atribuição do nomeAtributo

```
anotacaoAtributo = campo.getDeclaredAnnotation(NomeTagXml.class);
nomeAtributo = anotacaoAtributo == null ? campo.getName() : anotacaoAtributo.nome();
```

Framework Summer – Criando as anotações ControladorSummer e MapearRequisicao

- Vamos mudar agora criar nossas próprias classes de controle e mapeamento de requisição
- Para começar, vamos criar as anotações de Controlador e Mapeamento

```java
package br.com.diego.banco.anotacao;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD,ElementType.TYPE})
public @interface ControladorSummer {
public String value() default "";
}
```

```java
package br.com.diego.banco.anotacao;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.FIELD,ElementType.TYPE})
public @interface MapearRequisicao {
public String value();

}
```

Framework Summer – Criando as anotações ControladorSummer e MapearRequisicao

- Vamos criar uma classe que responderia com todos os caminhos das classes existentes em nosso projeto

```
package br.com.diego.summer.util;

import java.util.ArrayList;

public class CaminhoClasses extends ArrayList<String> {
private ArrayList<String> caminhoClasses = new ArrayList<String>();

public ArrayList<String> getCaminhoClasses(){
caminhoClasses = new ArrayList<String>();
caminhoClasses.add("br.com.diego.banco.dao.ContaDao");
caminhoClasses.add("br.com.diego.banco.Main");
caminhoClasses.add("br.com.diego.banco.controle.ContaController");
return caminhoClasses;

}
}
```

Framework Summer – Criando as anotações ControladorSummer e MapearRequisicao

- Criar um método em Summer para pegar o caminho

```
private String getCaminhoController() {
ArrayList<String> caminhosDasClasses = new CaminhoClasses().getCaminhoClasses();
MapearRequisicao anotacaoMapearRequisicao;
Class<?> classeControle;
try {
for (String caminhoClasse : caminhosDasClasses) {
classeControle = Class.forName(caminhoClasse);

if (classeControle.getDeclaredAnnotation(ControladorSummer.class) != null) {
anotacaoMapearRequisicao = classeControle.getDeclaredAnnotation(MapearRequisicao.class);

if (anotacaoMapearRequisicao != null && anotacaoMapearRequisicao.value().replaceAll("/", "")
.compareTo(this.request.getClasse()) == 0)
return caminhoClasse;
}
}
} catch (ClassNotFoundException e) {
e.printStackTrace();
throw new RuntimeException("Erro no caminho do controller");
}

return null;
}
```

Framework Summer – Criando as anotações ControladorSummer e MapearRequisicao

- Assim, nosso construtor em Summer ficaria somente assim

```
public Summer(String url) {
request = new Request(url);
this.caminhoClasse = getCaminhoController();
this.nomeMetodo = request.getMetodo();
}
```