* **Inversão de controle com injeção de dependência**

Ao invés de fazer

public class VendaDeProduto {

public void vendeProduto(Produto produto) {

//Todo o código para a venda do produto...

Log log = new Log("Arquivo.txt");

log.grava(produto);

}

}

Fazer:

public class VendaDeProduto {

private Log log;

public void VendaDeProduto(Log logVenda) {

this.log = logVenda;

}

public void vendeProduto(Produto produto) {

//Todo o código para a venda do produto...</span>

log.grava(produto);

}

}

* **Injeção de dependência**

Ao invés de fazer:

<?php

class Notification

{

public function send()

{

// TODO

}

}

class UserCommand

{

public function handle()

{

$notification = new Notification();

$notification->send();

}

}

Fazer:

<?php

class Notification

{

public function send()

{

// TODO

}

}

class UserCommand

{

protected $notification;

public function \_\_construct(Notification $notification)

{

$this->notification = $notification;

}

public function handle()

{

$this->notification->send();

}

}

* **Autowired**

Dispensa definir no applicationContext.xml a implementação da interface que será utilizada. Quando essa notação é utilizada, o Spring busca todas as implementações da interface e aplica a implementação que ele encontrar caso seja só uma, já inventando o valor do objeto criado, sem necessidade de criação do construtor pra injeção. Caso ache mais de uma implementação dessa interface, tem que usar o @Qualifier("nomeDaClasseDesejadaQueImplementaAInterface").

Exemplo sem autowired:

**private** FortuneService fortuneService;

**public** TrackCoach (FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}

E ainda precisamos definir no applicationContext.xml qual é a classe que implementa a interface FortuneService, dessa forma:

<bean id=*"myCoach"*

class=*"com.luv2code.springdemo.TrackCoach"*>

<constructor-arg ref=*"myFortuneService"*/>

</bean>

Agora, exemplo com autowired:

@Autowired

@Qualifier("randomFortuneService")

**private** FortuneService fortuneService;

Já esta usando o @Qualifier porque além da randomFortuneService, temos várias classes que implementam a interface FortuneService como, por exemplo, badFortuneService e happyFortuneService.

* **Prototype**

Primeiramente, temos que lembrar que todos os beans criados pelo Spring são Singleton. Logo, se crio duas instancias do mesmo bean, elas apontarão pro mesmo endereço de memória. Para criar duas instâncias do mesmo bean que apontam para endereços diferentes de memória, preciso adicionar a notação @Scope("prototype"). Por default, o Spring já vem com a notação @Scope("singleton").

Exemplo:

@Component

@Scope("singleton")

**public** **class** TennisCoach **implements** Coach{

…

}