**Nome: Marcio Montenegro de Sousa – Tokio Marine**

**Prova escrita**

**Orientação a objetos**

* O que é **injeção de dependência**?

R: É um Design Pattern ou um recurso que prega um tipo de controle externo, a um container, ou classe entre outros, que possibilita inserir uma dependência a fim de diminuir i acoplamento entre classes.

* O que você entende por **SOLID**?

R: Solid são cinco princípios da programação orientada a objetos que ajudam no desenvolvimento da aplicação. São eles

* 1. Princípio da responsabilidade única
  2. Princípio Aberto-Fechado
  3. Princípio da substituição de Liskov
  4. Princípio da Segregação da Interface
  5. Princípio da inversão da dependência
* Você conhece o design pattern **Strategy**? De um exemplo de quando utilizá-lo.

R: Pattern Strategy é um padrão conhecido de comportamento, sendo usado para gerenciar algoritmos, relacionamentos e responsabilidades entre objetos. Sendo o que define um conjunto de algoritmos que podem ser alterados para realizarem um comportamento específico.

A melhor forma de usar esse padrão é escolher um algoritmo a ser usado em tempo de execução, como por exemplo, para salvar arquivos.

Abaixo um exemplo de pattern Strategy usando Java com compactação de arquivos a onde criamos um zip.

// Interface da estratégia

interface pública CompessaoStrategy {

públicos vazios compresaoArquivos ( ArrayList < arquivo > arquivos );

}

E precisaremos fornecer nossas duas implementações, uma para zip e outra para rar

classe pública ZipCompressionStrategy implementa CompessaoStrategy {

públicos vazios compresaoArquivos ( ArrayList < arquivo > arquivos ) {

// usando a abordagem ZIP

}

}

classe pública RarCompressionStrategy implementa CompessaoStrategy {

públicos vazios compresaoArquivos ( ArrayList < arquivo > arquivos ) {

// usando abordagem RAR

}

}

**Spring Framework**

* Quais **subprojetos** da Spring você conhece? Descreva-os brevemente.

R: Tem o Spring MVC sendo uma estrutura orientada para HTTP baseada por Servlet e gerida pelo Spring Framework algo parecido com JSF em pilha JavaEE. Aonde os elementos mais popularessão as classes anotadas com @Controller para utilização de métodos que podem ser acessados usando diferentes solicitações HTTP e também tem o @Restcontroller equivalente para implementar APIs baseada e Rest.

Outro projeto seria o Spring Boot utilizada na configuração de aplicativos de uma forma mais rápida, disponibilizando configurações prontas para uso, afim de construir aplicativos em Spring.

* Como podemos criar e injetar **beans** no Spring?

R: Para o Beans, precisamos criar classes que serão utilizadas por outra classe como por exemplo a de serviço, abaixo segue o exemplo da classe PrimeiroBean, SegundoBean, TerceiroBean como sendo nosso Bean é a entidade **EntidadeService** possui a anotação@Autowired com os Benas para injetar essas classes de uma forma simples e limpa.

@**Component**

public PrimeiroBean {

...

}

@**Component**

public SegundoBean {

...

}

@**Component**

public TerceiroBean {

...

}

**public** **class** **EntidadeService** {

@Autowired

**private** PrimeiroBean primeiroBean;

@Autowired

**private** SegundoBeanr segundoBean;

@Autowired

**private** TerceiroBean terceiroBean;

*// método salvar*

}

* Como usar **transações** no Spring?

R: As transações devem ser entendidas como uma operação singular, quero dizer que sistemas que são gerenciados por banco de dados possui exatamente esse conceito de transação como, por exemplo, a anotação @Transactional que trabalha dentro do escopo do banco de dados ou salva os dados no banco de dados eu não salva.

**Rest**

* O que você entende sobre o que são serviços web RESTful?

R: É a capacidade de uma determinada aplicação utilizar os princípios REST, que é uma abstração a arquitetura Web, resumindo consistem em princípios, regras e constraints que, quanto seguidas, permitem a criação de um projeto com interfaces definidas.

* Cite algumas boas práticas envolvendo rest?

R: A primeira seria a o conhecimento na fundamentação básica dos métodos Get/Post/Put/ Patch/Delete sem isso é a mesma coisa da pessoas ser evangelia e nunca ter lido a bíblia.

A Segunda seria tratar as Pais como se fossem produto de negocio, para isso precia aproxima a equipe de TI com a de negócio.

A terceira seria criar APIS Robustas para não exponha funções de negócios, devem possuir testes, consistentes entre times de TI, documentação.

* Cite os métodos HTTP mais usados na arquitetura baseada REST e o que eles fazem?

R: Abaixo os métodos mais usados e o que fazem

* Método GET - método seguro – idempotente, usado quando precisa de uma representação do recurso.
* Método Post - usado para criar um novo recurso / enviamos um corpo de informações para que seja salva, ele não é idempotente.
* Método PUT – serve para atualizar informações existentes, para isso tem que passar um parâmetro exemplo(Id) e enviamos um corpo de requisição para alteração a informação tem que ser passada completa, senão pode entrar nulo - é um método idempotente.
* Método Patch - serve para alterar um recurso parcialmente, podendo passar apenas um atributo e altera somente ele e não os outros. - é um método idempotente
* Metodo Delete – é um recuso que exclui informações através do parâmetro que é passado como o Id, por exemplo.
* Qual a diferença entre os códigos de resposta (status codes) HTTP 200, 400 e 500?

R: São Status do HTTP do tipo de resultado.

* No caso do 200 tratasse de status , sucesso bem sucedido ou atendimento na criação ou inclusão, ou serviço que foi processado com sucesso, entre outros.
* Já o 400 se refereca Requisição mal feitas, erro na descrição - Indica ERRO por parte do consumidor da API.
* Status 500 erro interno e inesperado, responsabilidade no nosso código, que está com erro.

**Prova prática**

1. Acesse <https://github.com/leonardohenrique/tokio-test.git> e clone o projeto.
2. Importe o projeto no seu Eclipse
3. Rode a aplicação Spring Boot e acesse <http://localhost:8080/customers> (deverá exibir uma lista de clientes).
4. Abra a classe **com.example.api.web.rest.CustomerController** e adicione **endpoints** para criar um novo cliente, editar um cliente e excluir um cliente.
5. Valide os dados antes de cadastrar ou editar
6. Pagine a listagem de clientes.
7. Possibilite o cadastro de múltiplos endereços para um cliente.
8. No cadastro de endereço permita inserir apenas o CEP carregando os dados via consumo do serviço: <https://viacep.com.br/>
9. Envie a url do seu repositório no github para análise.

Obs.: Será um diferencial implementações como: tratamento de exceções (RestControllerAdvice), testes, validações, frontend, autenticação e documentação.