Trabalho 4 - Programação Orientada a Objetos Avançada
UFSCar - ENPE 2020.1 - Bloco B
Prof. Daniel Lucrédio

Autores: João Victor Mendes Freire, 758943
Julia Cinel Chagas, 759314

1. **Quais são as três principais razões de um projeto ruim de software? Explique cada uma delas. (valor = 0,5)**

   Segundo o autor Robert Martin [3], existem três características que fazem com que um projeto de software seja considerado ruim: Rigidez, Fragilidade e Imobilidade.

   Rigidez diz respeito a um sistema em que mudanças são desencorajadas, uma vez que para realizar uma é necessário alterar diversos módulos. Isso é observável em códigos que não são devidamente modularizados. Considere uma aplicação que, em diversos momentos, verifica se um dado é válido, e se não for, imprime um erro na saída e encerra a execução. Uma mudança para escrever o erro num arquivo de *logs* ou então deixar de encerrar a execução e imprimir o erro apenas como um aviso, exigiria procurar e alterar por todo o sistema onde essa funcionalidade é implementada.

   Um sistema frágil é aquele em que, ao realizar uma modificação, partes inesperadas do código deixam de funcionar. Também é característico de uma modularização mal feita.

   Por fim, imobilidade é uma característica de sistemas que não podem ser reutilizados em aplicações diferentes devido ao acoplamento com o ambiente para os quais foram desenvolvidos. Isso sugere uma falta de abstrações no código, fazendo com que ele fique dependendo de detalhes específicos do contexto para o qual foi desenvolvido.

2. **O que é o princípio da inversão de dependência? (valor = 0,5)**

   O *Dependency Inversion Principle* (DIP) é o princípio SOLID que diz que módulos de Alto Nível não devem depender de módulos de Baixo Nível, mas ambos devem depender de abstrações. E abstrações não devem depender de detalhes, mas detalhes devem depender de abstrações. De forma bastante complementar ao OCP e ao LSP, o princípio reforça a necessidade de abstrações para manter o código reutilizável, modular e extensível.

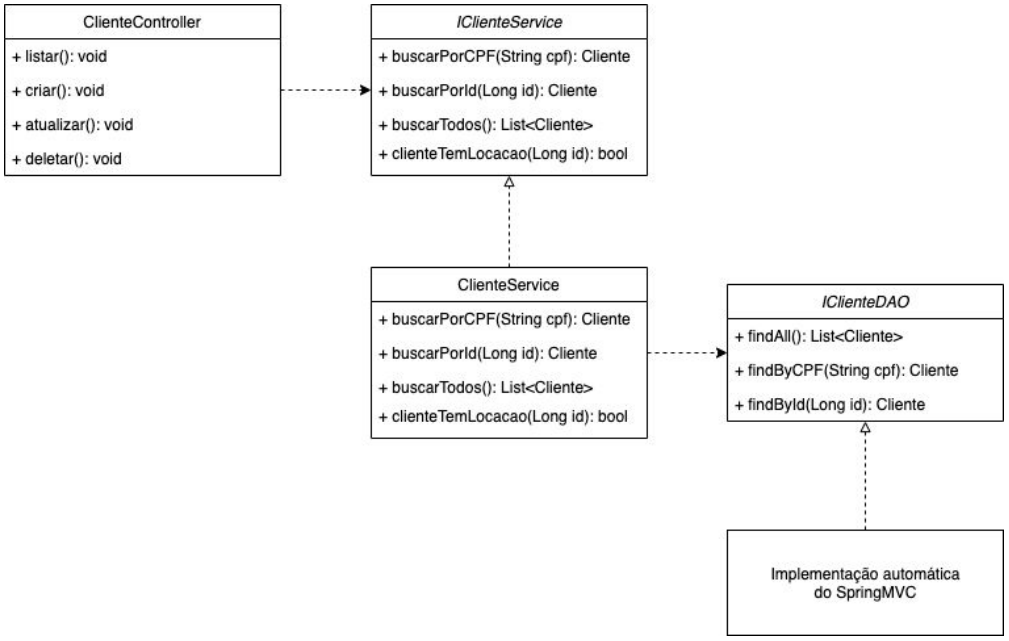
3. **Por que simplesmente estruturar um software em camadas não é suficiente para se conseguir um bom design? (valor = 1,0)**

   Porque pode causar dependência transitiva, já que classes criadas nesse modelo de camadas irão depender da classe pertencente à camada inicial, fazendo com que uma mudança nesta primeira camada prejudique todas as que estiverem ligadas entre si.

4. **Dê um exemplo simples (diferente daqueles mostrados nos artigos) da aplicação errada/certa do princípio da inversão de dependência. Use trechos de código/diagramas para ilustrar. (valor = 1,5)**

O exemplo a seguir mostra um uso correto segundo o princípio da inversão de dependência. O exemplo foi tirado de um trabalho da disciplina Desenvolvimento de Software para Web 1. No trecho em questão, o controlador do CRUD Cliente tem um atributo do tipo `IClienteService`, que é uma interface. A implementação dessa classe é isolada, e portanto, não importa para o controlador a origem dos dados.

A implementação apresentada (`ClienteService`) recupera os dados de um banco de dados MySQL criado automaticamente com Spring Data JPA, assim, utiliza uma interface `IClienteDAO` para indicar as operações. Quem implementa essa interface é o próprio Spring. Um uso incorreto seria o controlador do Cliente usar o Service diretamente, deixando o código preso a implementação que recupera a informação do banco de dados.



5. **O que é inversão de controle? (valor = 0,5)**

Tradicionalmente, ao utilizar bibliotecas um usuário tem o controle do código. É ele que faz as chamadas para as funções e classes disponíveis. Em frameworks o que acontece é que o usuário informa o que deve ser chamado em certo contexto, e quem controla a execução e faz essas chamadas é o framework.

Assim, existe uma inversão em quem está no controle: no primeiro quem controlava a biblioteca era o usuário, e no segundo quem invoca os códigos do usuário é o framework. Essa dinâmica é conhecida exatamente como Inversão de Controle, ou então Princípio de Hollywood.

6. **O que é injeção de dependência? (valor = 0,5)**

   A Injeção de Dependência é um padrão de projeto utilizado para implementar a Inversão de Controle em um sistema. Nesse padrão existe um montador responsável por preencher a implementação adequada das interfaces utilizadas, seja passando como argumento em um construtor, através de um método setter ou através de uma interface.

7. **Qual a relação entre inversão de dependência, inversão de controle e injeção de dependência? (valor = 1,5)**

   A inversão de dependência é um princípio SOLID cujo objetivo é fazer com que outras classes dependam de classes abstratas, tornando-as desacopladas. A injeção de dependência é uma maneira de seguir esse princípio e, por outro lado, também é uma forma de realizar a inversão de controle.

8. **O que é um "Service Locator" e qual a sua diferença em relação à injeção de dependência? (valor = 1,5)**

   A ideia de um Service Locator é ser um objeto que recupera todos os serviços que uma aplicação pode precisar. Ambos Service Locator e Injeção de Dependências são bastante similares. A principal diferença entre eles se dá no fato de que no Service Locator a própria classe cria as suas dependências, utilizando o Locator para isso. Já na Injeção de Dependências a classe recebe de fora os objetos.

9. **Como é possível usar o princípio da segregação de interfaces para um uso melhor do padrão "Service Locator"? (valor = 1,5)**

   Ao utilizar o princípio da segregação de interfaces, que diz que um objeto não deve depender de uma interface que ele não precisa, evita-se que uma classe tenha acesso a demais métodos do Service Locator que não foram originalmente planejados para ela. O uso desses outros métodos pode gerar comportamentos inesperados caso alterações sejam feitas nessas funções (já que o módulo não é considerado dentre os que utilizam tal método).

10. **Pesquise e cite alguns frameworks modernos de desenvolvimento de software (em qualquer linguagem) que utilizam os conceitos abordados aqui. (valor = 1,0)**

    O framework Angular.js de JavaScript, o .NET de C# e o Spring de Java são alguns frameworks que implementam Injeção de Dependências. Um framework que implementa Service Location é o Yii de PHP. O Dagger 2, um framework Android, e o Avalon, outro framework Java, implementam ambos.

Referências Bibliográficas

[1] FOWLER, Martin. **Inversion of Control**. martinfowler.com, 2005. Disponível em <https://martinfowler.com/bliki/InversionOfControl.html>. Acesso em 15/01/2021.

[2] FOWLER, Martin. **Inversion of Control Containers and the Dependency Injection pattern**. martinfowler.com, 2004. <https://martinfowler.com/articles/injection.html> Acesso em 15/01/2021

[3] MARTIN, Robert. **The Dependency Inversion Principle**. Object Mentor SOLID Design Papers, 1996.