

CETEJ35 - Java Web - JAVA_XXX (2024_01)

[Meus cursos](#) / [CETEJ35 - Web \(2024_01\)](#) / [Semana 06: 07/10 a 13/10](#) / [Integração](#)

Integração

✓ **Feito:** Ver

A fazer: Gastar pelo menos 20 minutos na atividade

A fazer: Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Normalmente, um aplicativo real vai precisar mais do que salvar dados em memória - ele vai precisar de persistência em banco! É aí que entra o conteúdo dessa aula.

ARQUITETURA

Segundo Bass, Clements & Kazman, arquitetura é o conjunto de estruturas que permitem analisar o software. Essas estruturas são formadas por elementos de software e relações entre eles, além das propriedades de ambos [1].

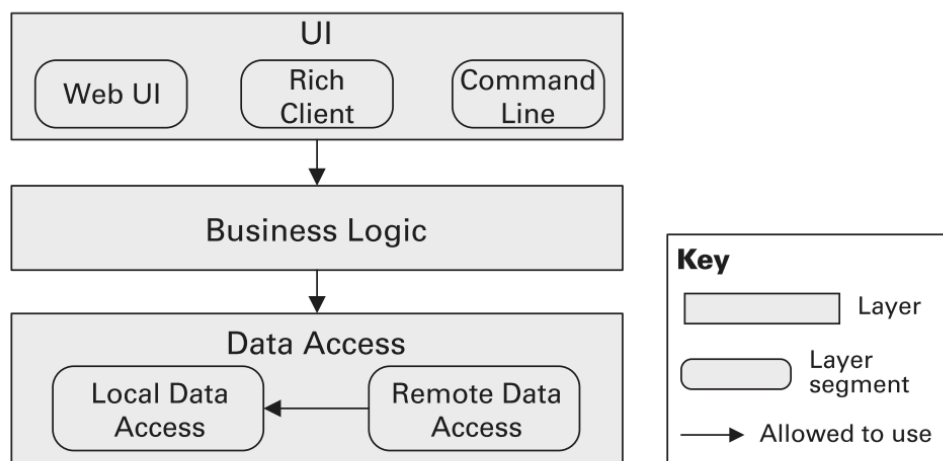
Existem vários tipos de estruturas. Dessa forma, quando falamos de arquitetura de software não estamos nos referindo a essas estruturas [1]. Por exemplo, o conjunto de classes de um aplicativo em Java forma uma estrutura arquitetural chamada de estrutura de classes [1].



Uma das estruturas comuns na arquitetura de software é a estrutura de camadas. Você já deve ter ouvido falar nisso. Estruturar em camadas significa agrupar partes da aplicação de acordo com o propósito geral dessas partes. Por exemplo, você pode criar uma camada de persistência agrupando em um mesmo pacote as classes responsáveis por salvar os dados no banco de dados. Também poderia ter classes de serviço, que seriam responsáveis pelas regras de negócio. No caso da nossa aplicação, poderíamos colocar a classe **CidadeRepository** em um pacote chamado persistencia.

A Figura abaixo (retirada de [2]) mostra um exemplo comum de arquitetura em camadas. Aplicações empresariais frequentemente são criadas usando esse tipo de estrutura. Assim, temos divisões claras entre as responsabilidades. A principal vantagem desse tipo de estrutura é que ela permite a independência das camadas. Assim, pelo menos teoricamente, você poderia substituir uma camada sem afetar as demais. Isso favorece a portabilidade e manutenibilidade.

Figure 2.20
Layered design with segmented layers



Contudo, a medida que o aplicativo cresce, fica cada mais difícil definir onde cada classe deveria ser colocada. É comum começarem camadas de utilidades onde são colocadas todas as classes que não se encaixam em outras camadas. Isso frequentemente diminui a manutenibilidade da aplicação e impacta de forma negativa o acoplamento.

Por isso, diversos profissionais do desenvolvimento tem defendido o uso de uma estrutura orientada a funcionalidade, em vez de propósito [3]. Por exemplo, no caso da nossa aplicação teríamos um agrupamento das classes que compõem a funcionalidade de gerenciamento de cidades, independentemente se o propósito da classe é definir as regras de negócio ou fazer a persistência.

Agrupar as classes por funcionalidade facilita a modularização da aplicação, uma vez que todas as classes de uma mesma funcionalidade estão reunidas em um só lugar.



Atualmente, nossa aplicação tem um só pacote, **visao**, onde estão todas nossas classes. Inicialmente, usamos esse nome para agrupar as classes relacionadas ao padrão MVC, porque estávamos pensando em camadas. Contudo, nossa aplicação cresceu e agora tem uma classes para persistência de dados em banco. Por isso, agora vemos que esse não é um bom nome, pois não representa bem a funcionalidade que estamos agrupando ali.

Por isso, vamos alterar o nome do pacote para **cidade**, em vez de **visao**.

Observe que ao mudar o nome de um pacote você também precisa alterar a instrução **package**, dentro da classe, que define o pacote no qual a classe está. Normalmente, as IDEs fazem essa mudança automaticamente quando você muda o nome do pacote.

[1] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3rd ed. Boston, Massachusetts: Addison-Wesley, 2013.

[2] P. Clements et al., Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2003.

[3] V. Vernon, Implementing Domain-Driven Design. Addison-Wesley Professional, 2013.

O código desenvolvido nesta Seção está disponível no [Github](#), na branch **semana05-20-integracao-funcionalidade**.



Retroceder

Avançar

[◀ Verificação de aprendizado - Validação de Dados](#)

Seguir para...

[API Reativa ▶](#)



 [Contate o suporte do site](#) 

Você acessou como RAFAEL ROCHA DA SILVA PROENCA (Sair)
CETEJ35 - Web (2024_01)

- Tema
- Adaptable
- Boost
- Clássico
- Campus
- Apucarana
- Campo Mourão
- Cornélio Procópio
- Curitiba
- Dois Vizinhos
- Francisco Beltrão
- Guarapuava
- Londrina
- Medianeira
- Pato Branco
- Ponta Grossa
- Reitoria
- Santa Helena
- Toledo
- UTFPR
- Ajuda
- Chat UTFPR
- Calendário Acadêmico
- Biblioteca
- e-Mail
- Nuvem (OwnCloud)
- Produção Acadêmica
- Secretaria Acadêmica
- Sistemas Corporativos
- Sistema Eletrônico de Informação - SEI
- Suporte ao usuário
- Criação de curso
- Comunidade
- Português - Brasil (pt_br)
- Deutsch (de)
- English (en)
- Português - Brasil (pt_br)

Resumo de retenção de dados

Baixar o aplicativo móvel.

 [Dê um feedback sobre este software](#) 

Universidade Tecnológica Federal do Paraná - UTFPR
Suporte ao usuário

