Pós-Graduação
Engenharia de Software

Arquitetura de Software e Padrões de Projeto

Aula 02

Prof. Msc Rogério Augusto Rondini
rarondini.paradygma@gmail.com

Conteúdo

- Estilos Arquiteturais
- Arquitetura em Camadas
- Revisão de conceitos
  - Responsabilidades
  - Acoplamento e coesão
- Introdução a *Design Patterns*
- O padrão MVC
- Aplicação no Estudo de Caso

Conceitos

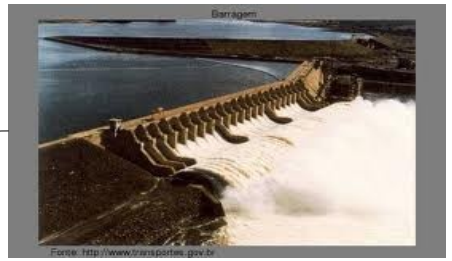
- O que é Acoplamento ?
- O que é Coesão ?
- O que é encapsulamento ?

Encapsulamento

- Manter oculta algumas informações sobre um determinado objeto
- Separação de um sistema/programa em partes, ocultando detalhes de implementação

Encapsulamento

Para usar energia elétrica,
não precisamos conhecer detalhes
do processo de geração e distribuição.



Em desenvolvimento de software orientado a objetos, podemos aplicar a mesma ideia

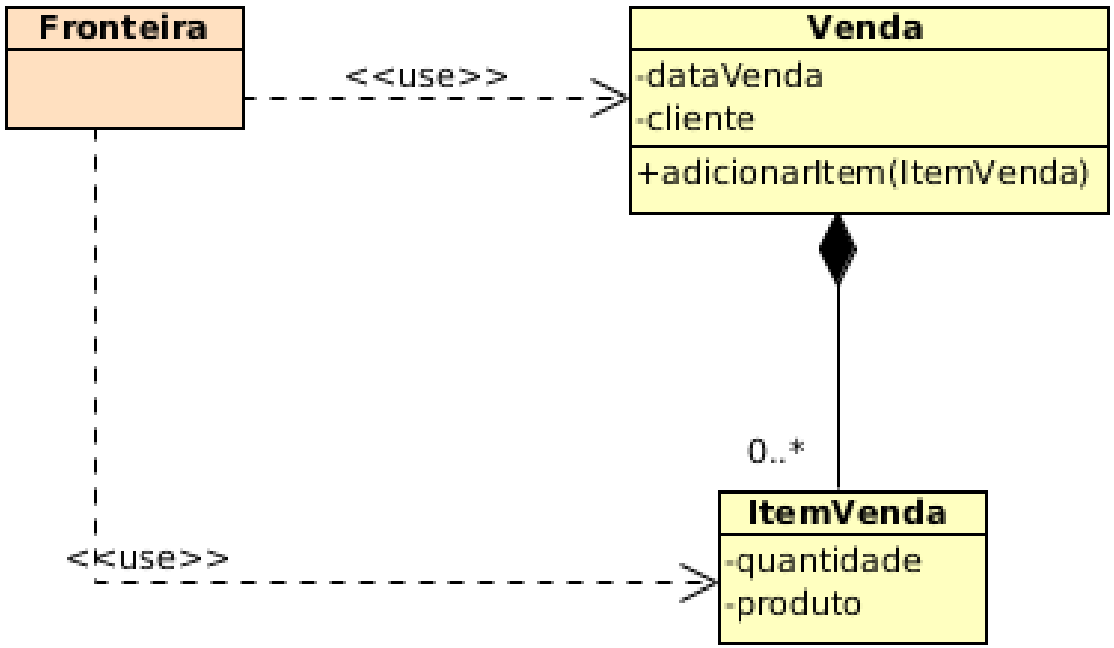
Acoplamento/Coesão

- Acoplamento
  - Grau de relacionamento entre classes, ou seja, o quanto uma classe conhece de outras classes
- Coesão
  - Nível de responsabilidade de uma classe, ou seja, o quanto uma classe é específica para realizar tarefas em um determinado contexto.

Em O.O, o objetivo é diminuir o acoplamento aumentar a coesão.

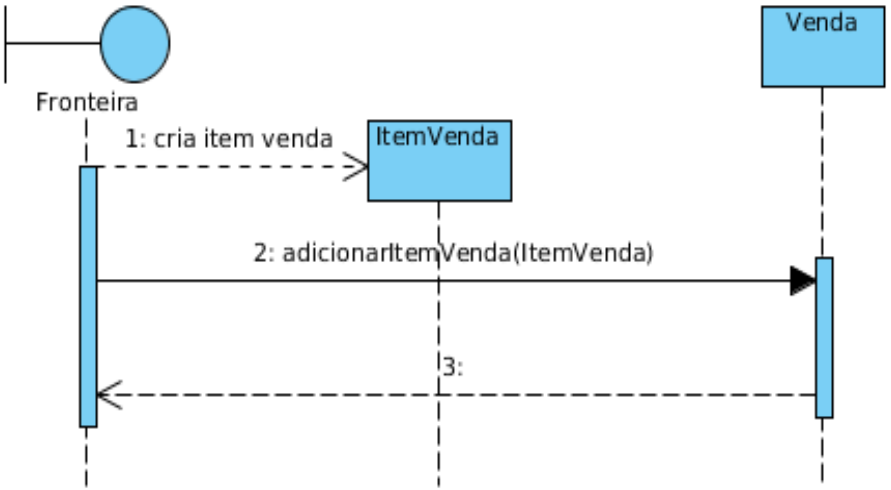
Exemplo

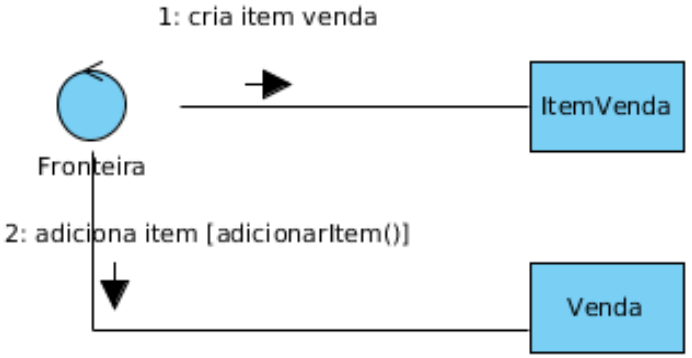
- O diagrama de classes a seguir ilustra um cenário de inclusão de Item de Venda em uma Venda
- Qual o grau de acoplamento da classe “Fronteira” ?



Exemplo (cont.)

- Agora, os diagrama de sequência e comunicação mostrando o fluxo de execução





Exemplo (cont.)

- Como implementar

```
class Venda {
    ItemVenda[] itens;

    método adicionarItem (ItemVenda iv) {
        incluir iv em itens;
    }
}
```

"Venda" possui dependência com "ItemVenda", pois já existe uma associação entre elas

```
class Fronteira {
    …
    Venda v
    …
    ItemVenda novoItem
(1) Cria instância de ItemVenda e atribuir a novoItem

(2) v.adicionarItem(novoItem)

}
```

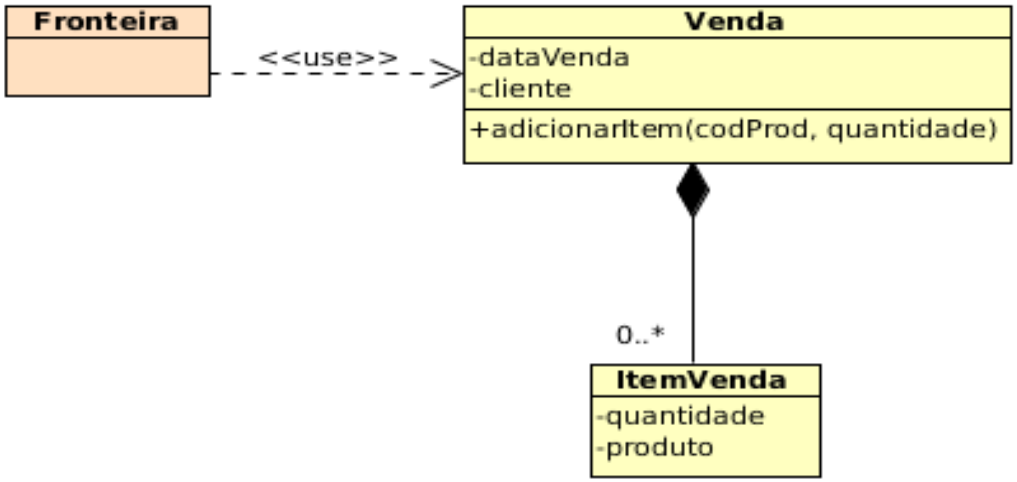
Se "Fronteira" for responsável por criar instância de "ItemVenda", também haverá uma dependência

Exemplo (cont.)

- É possível observar que a classe “Fronteira” está acoplada tanto à classe “Venda” quanto à classe “ItemVenda”
- **Como reduzir o grau de acoplamento da classe “Fronteira” ?**

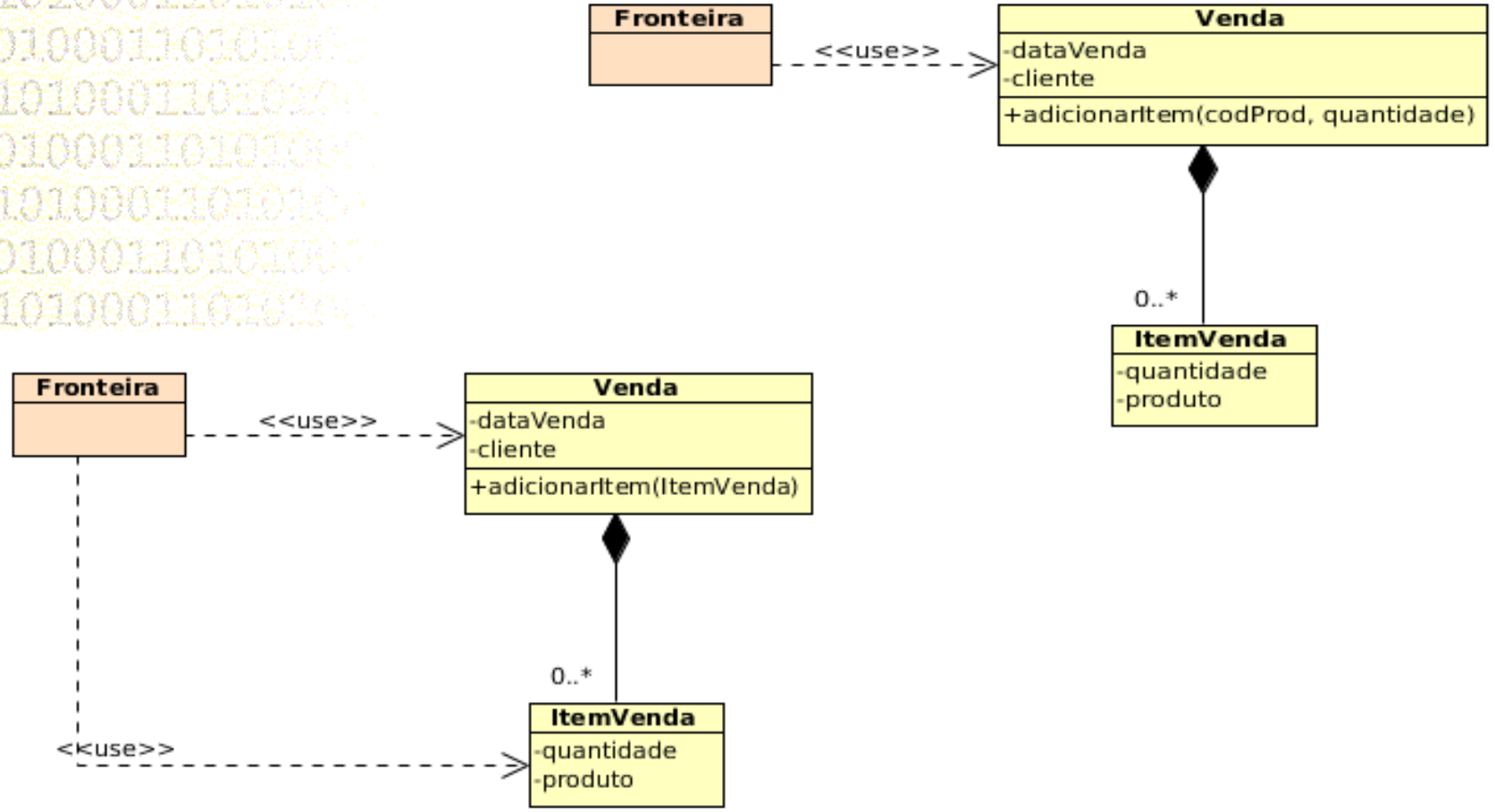
Exemplo (cont.)

- Mesmo cenário, porém reduzindo acoplamento da classe Fronteira



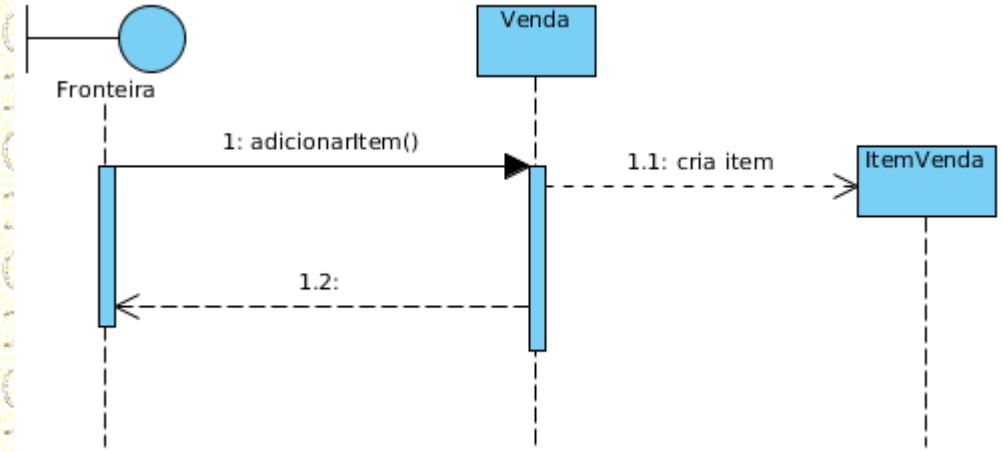
Exemplo (cont.)

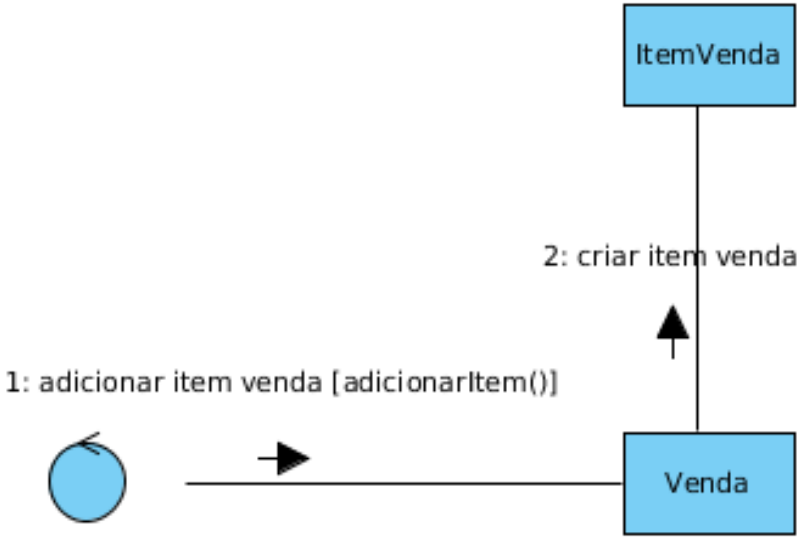
- Comparação



Exemplo (cont.)

- Agora, os diagrama de sequência e comunicação mostrando o fluxo de execução após modificação





Exemplo (cont.)

- Como implementar após alteração

```
    class Venda {
        ItemVenda[] itens;

        Método adicionarItem (codProd, quantidade) {
            ItemVenda novoItem
(1.1)       Cria instância de ItemVenda passando os parâmetros
                codProd e quantidade, e atribui a novoItem
            incluir novoItem em itens;
    }
}
```

```
    class Fronteira {
        …
        Venda v
            …
(1)     v.adicionarItem(codProd, quantidade)

    }
```

"Fronteira" não tem mais dependência com "ItemVenda".

Se "Venda" for responsável por criar instância de "ItemVenda", não existirá nenhuma dependência adicional

Exemplo (cont.)

- Com a alteração
  - Classe “Fronteira” não precisa depender da classe “ItemVenda”, pois não está mais criando instância de “ItemVenda”.
  - Classe “Venda” já tinha dependência com “ItemVenda” em função da associação (composição), portanto, o fato de “Venda” passar a ter a responsabilidade de criar instância de “ItemVenda”, não acrescenta nenhuma dependência nova.

Estilos Arquiteturais

Estilos Arquiteturais

- Estilo arquitetural pode ser considerado um conjunto de princípios e padrões que direcionam a arquitetura de um software
- Em geral, estilos arquiteturais podem ser classificados quanto
  - comunicação
  - implantação
  - Estrutura
  - Interação

Estilos Arquiteturais

- Comunicação
  - Mensagem, SOA...
- Implantação
  - **Cliente/Servidor, 3-camadas...**
- Estrutura
  - Componentes, Objetos, **Camadas**...
- Interação
  - **Separação da apresentação** (ex. tela)

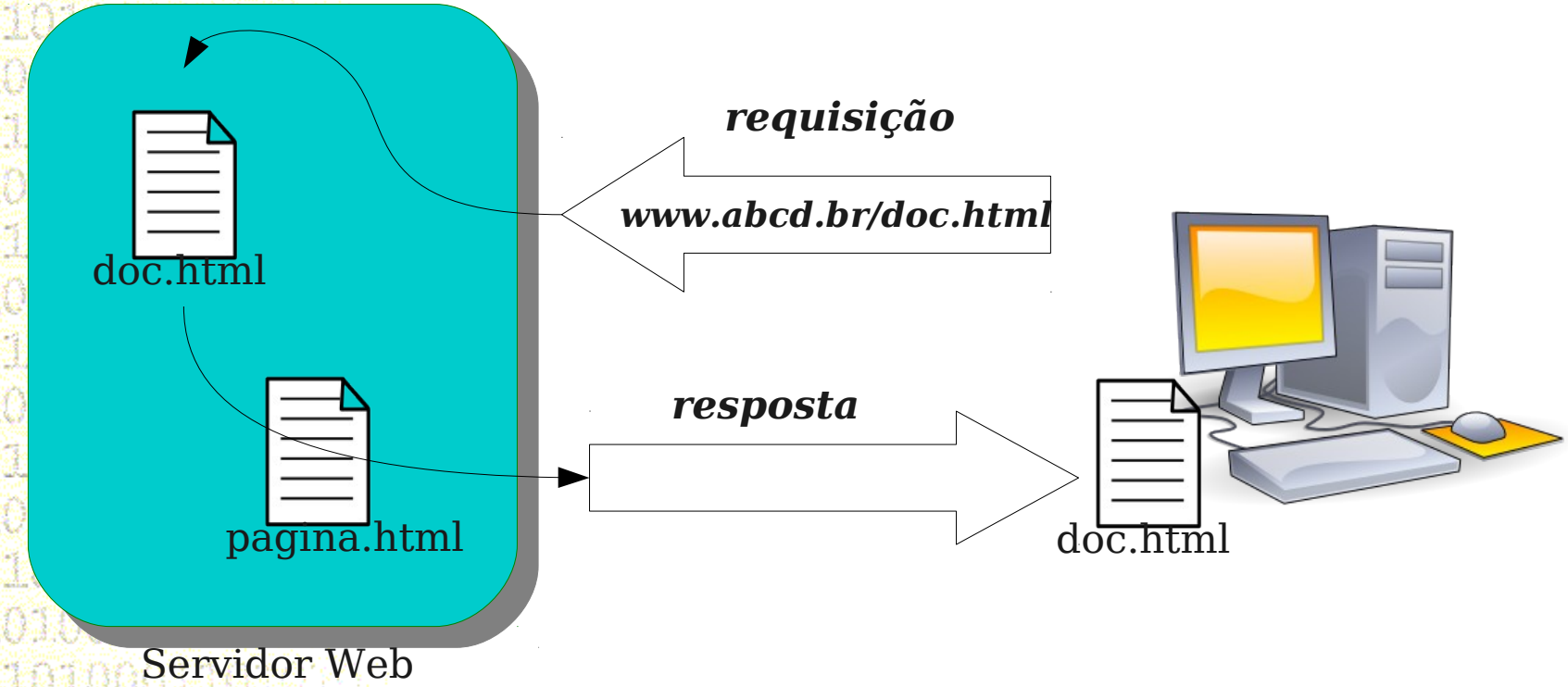
Estilos Arquiteturais

- Os estilos arquiteturais podem ser combinados para formar a arquitetura de um software, por exemplo...
  - Uma aplicação Web poderá ser separada em camada de apresentação e camada de negócio (camadas).
  - As camadas podem ser implantadas em diferentes servidores, um para camada de apresentação e outro para camada de negócio, além do banco de dados (implantação 3-camadas)

Cliente Servidor

- Arquitetura Cliente/Servidor, como o próprio nome diz, separa a implantação em dois grandes componentes, aplicação cliente e aplicação Servidora
  - Aplicações Web são consideradas aplicações Cliente/Servidor, onde
    - Cliente → navegador Web
    - Servidor → servidor Web

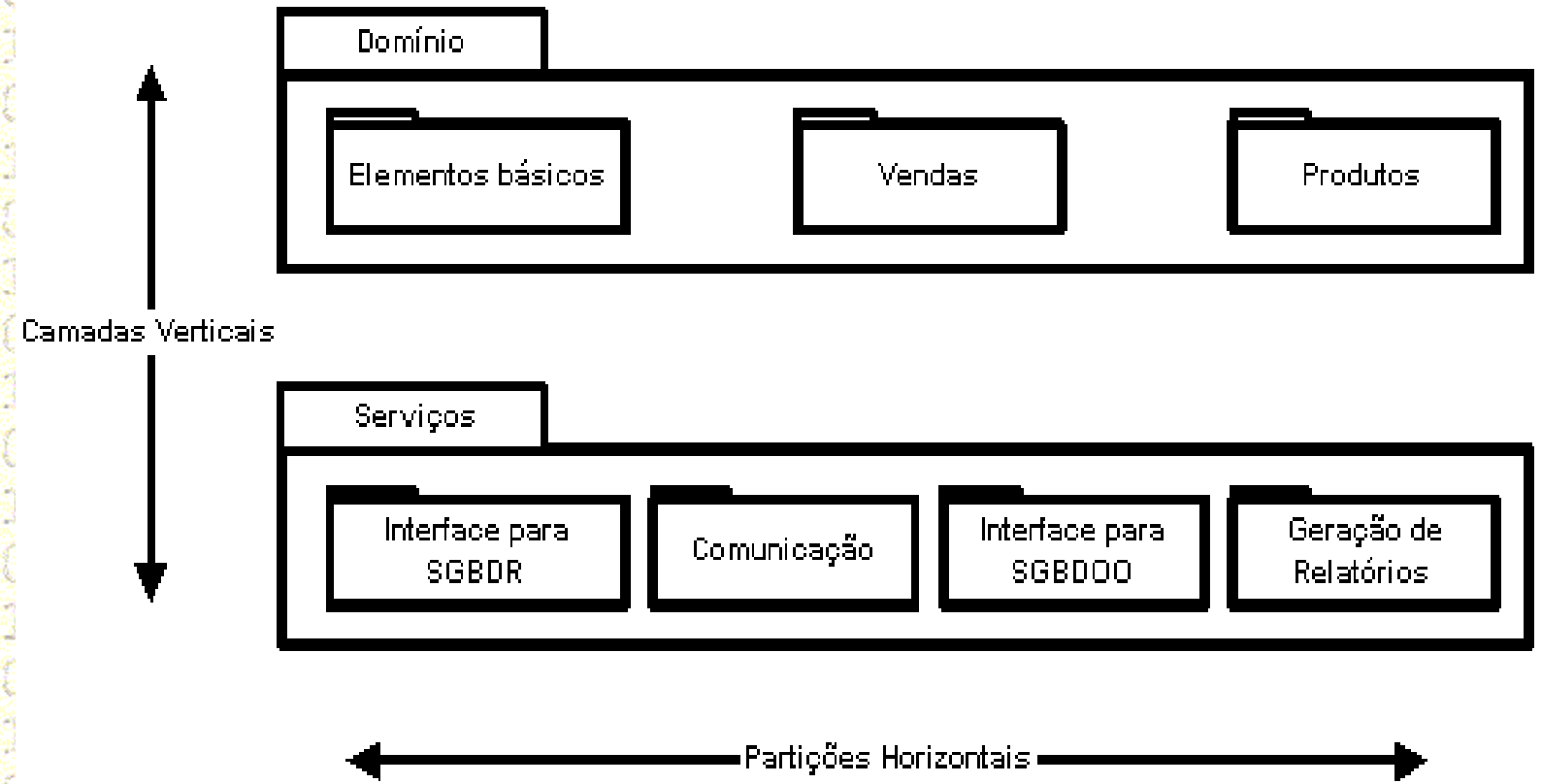
Cliente Servidor



Layers (camadas)

- Um dos principais estilos arquiteturais utilizados atualmente
- Utilizado em conjunto com diversos outros estilos
- O principal objetivo é decompor um sistema em “camadas” e “partições” lógicas, com responsabilidades e interfaces bem definidas.

Layers (camadas)



Layers (camadas)

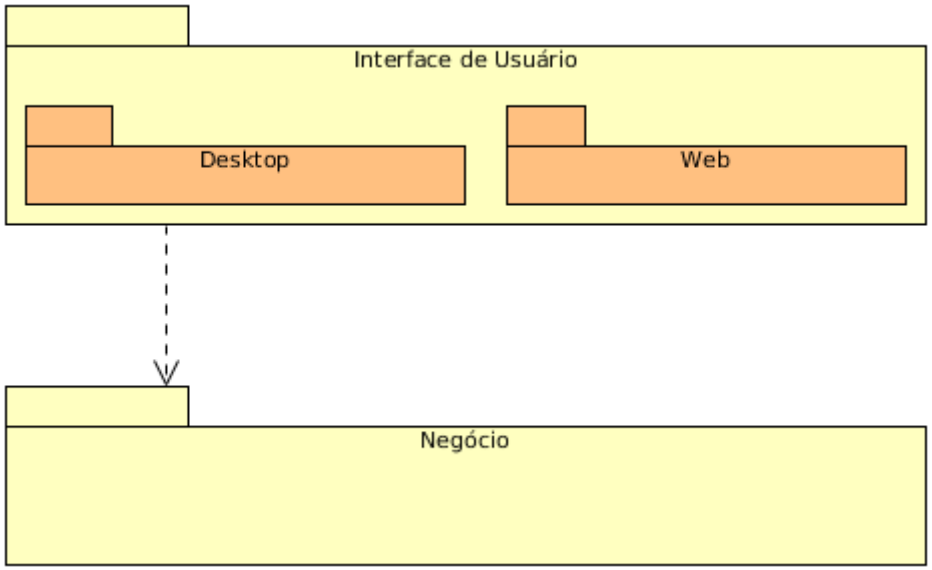
- Acesso através de interface entre camadas através de interface
- Camada A não conhece detalhes de implementação da camada B
- Usa princípio do encapsulamento
- Mantém o acoplamento fraco
- Uso de padrões de projeto para organização das camadas
- Dependência de “cima” para “baixo”

Layers (camadas)

- Vantagens
  - Padronização
  - Reuso
  - Troca de implementações
- Desvantagens
  - Aumenta complexidade de Design

Separação de Apresentação

- É um tipo de arquitetura em Camadas, onde o objetivo é separar detalhes da apresentação (telas) dos detalhes de processamento



Atividade - Sala

- Quais estilos arquiteturais se encaixam no sistema PDV ?
- Como podemos resolver a necessidade de implantação do PDV nos terminais caixas do supermercado e futuramente como uma aplicação Web
  - Descreva a solução através de um diagrama de pacotes ou componentes.

Introdução – Patterns

Patterns

- Solução reutilizável
- Modelo para implementação de soluções
- Descrição nomeada de um problema acompanhado de uma solução que pode ser aplicada em contextos diferentes
- Vocabulário comum para facilitar a compreensão de soluções entre desenvolvedores

Patterns

- A noção de padrões surgiu originalmente em meados dos anos 70 na construção civil através do arquiteto Christopher Alexander
- Na engenharia de software, a ideia de descrição de padrões surgiu em meados dos anos 80 através de Kent Beck
- Em1994 foi publicado o livro “Design Patterns Elements of Reusable Object-Oriented Software” por Gamma, Helm, Jo hnson e Vlissides (*GoF – Gang of Four*)

Patterns

- devem ter nomes sugestivos
  - Estimula agrupamento de ideias em um conceito
  - Incorpora tal conceito à nossa compreensão
  - Facilita a comunicação
- Abstração
  - Nomear ideia complexa em uma idéia simples eliminando-se detalhes

Patterns

- Exemplo
  - Nome: *Singleton*
  - Problema: O sistema necessita de um único ponto de acesso. É permitida apenas uma instância do objeto
  - Solução: Defina um método em uma classe que retorne apenas uma instância da classe e não permita à classe ser instanciada externamente
- Curiosidade
  - *Linda Rising* (notável na área de engenharia de software) documentou mais e 500 patterns em seu livro “Pattern Almanac 2000” (http://www.lindarising.org/)

Padrão MVC

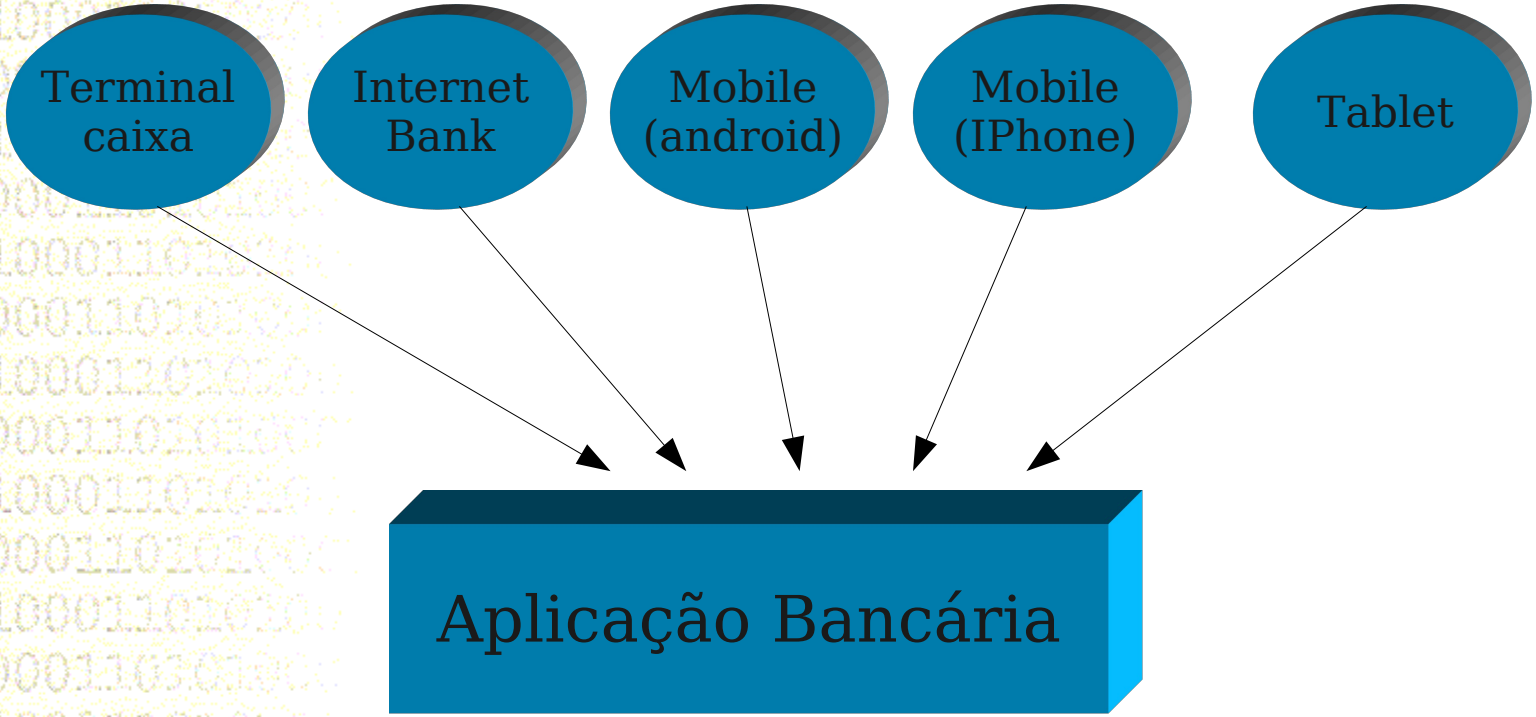
Model, View e Controller

MVC

- Padrão Arquitetural que obedece ao estilo arquitetural “Separação de Apresentação”
- Divide a aplicação em três partes distintas, com responsabilidades bem definidas
  - Model → lógica de negócio
  - View → apresentação
  - Controller → fluxo de navegação

MVC

- Problema

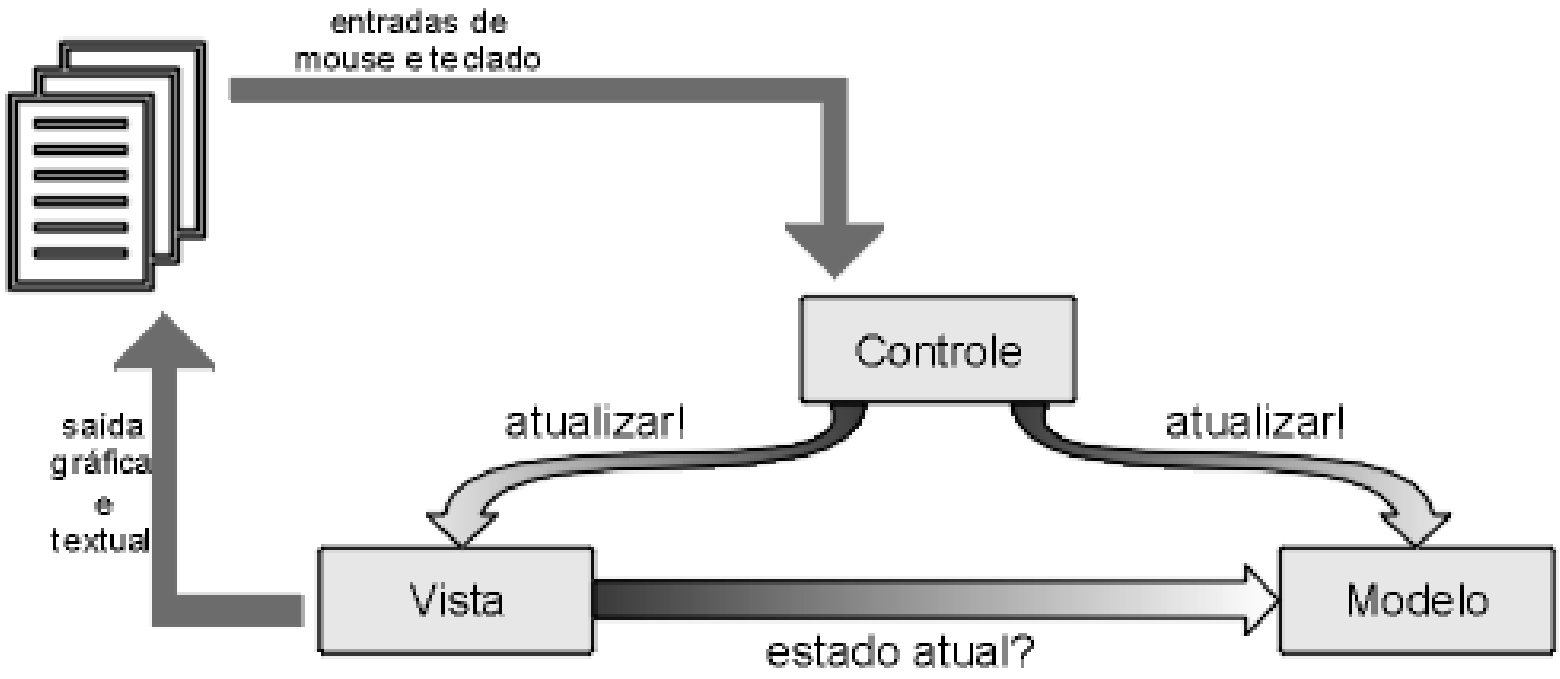


MVC

- Solução
  - Separação da lógica de negócio (cálculos, regras, acesso a dados), da lógica de apresentação
- No exemplo anterior...
  - Apresentação muda para cada tipo de dispositivo, mas as regras bancárias permanecem as mesmas

MVC

- Como funciona ?
  - Importante lembrar das responsabilidades



MVC

- Como funciona – ***Model***
  - Representa os dados corporativos e as regras de negócio que governam o acesso e as atualizações dos dados
  - Camada representando o núcleo do sistema
    - Calcular imposto
    - Extrato mensal
    - Transferência (DOC/TED)

MVC

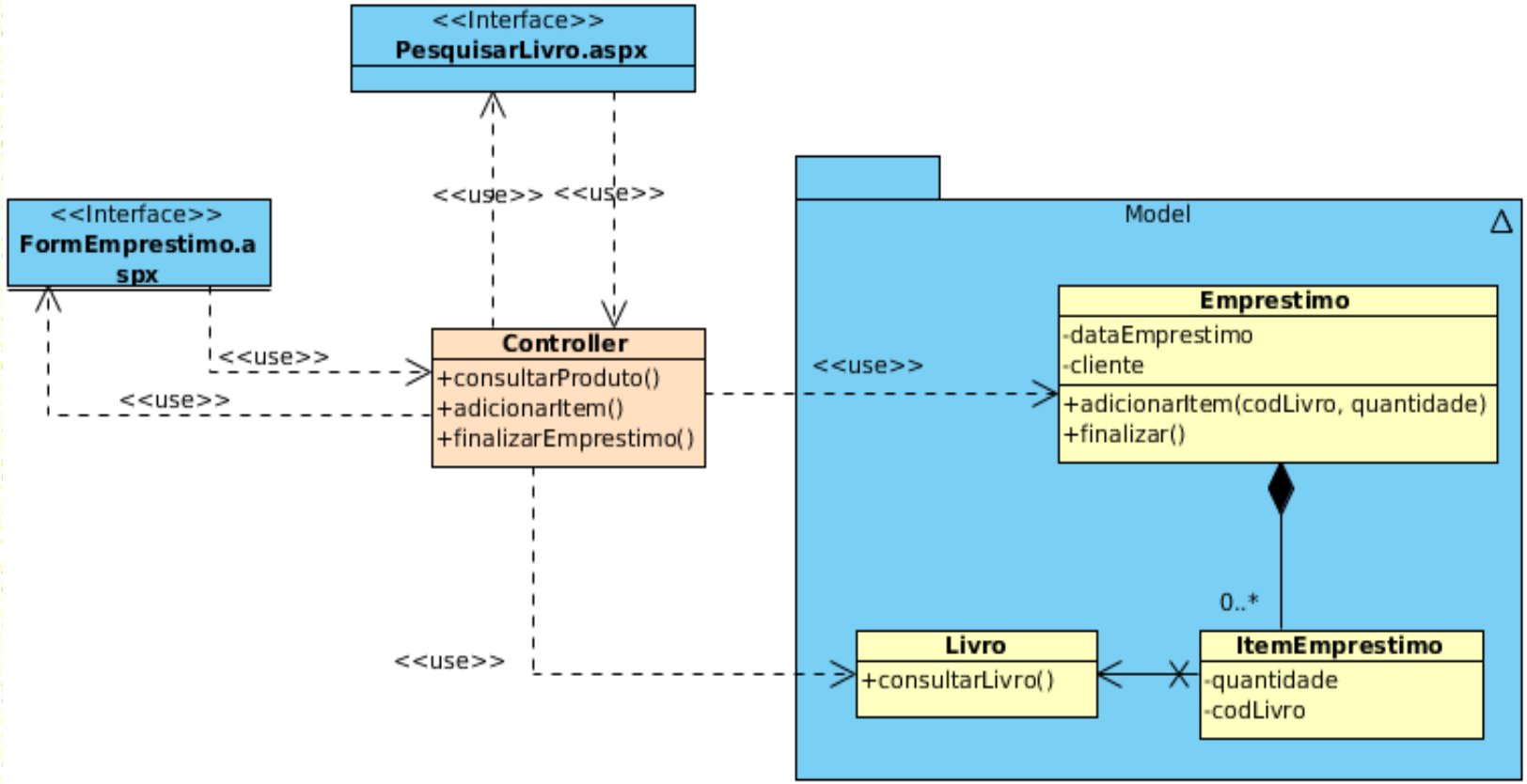
- Como funciona – ***View***
  - Exibe o conteúdo/resultado do processamento da Model.
  - Define como os dados serão exibidos.
    - Compatibilidade entre navegadores
    - Apresentação em dispositivos móveis
    - ...

MVC

- Como funciona – ***Controller***
  - Traduz interações do usuário com a *View* em ações a serem executadas pelo *Model*.
    - Clique em botões em aplicações Desktop
    - Requisições HTTP em aplicações Web
  - Baseado na interação do usuário e na resposta do *Model*, o *Controller* responde direcionando para a View apropriada.

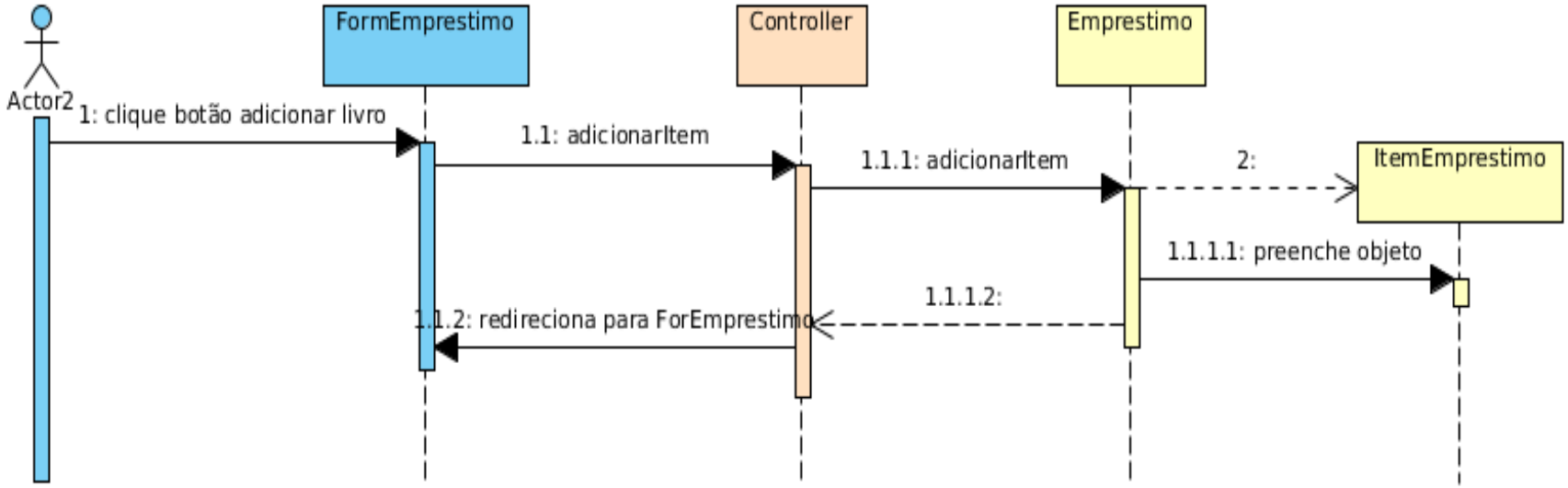
MVC

- Exemplo – empréstimo de livros



MVC

- Exemplo – empréstimo de livros
  - Diagrama de sequência para adicionar livro



MVC

- Em algumas situações, é possível trocar a *View* sem trocar o *Controller*
  - Aplicações Web escritas na mesma plataforma, porém com apresentação diferente para cada tipo de navegador
- Em algumas situações, deve-se trocar também o *Controller*
  - Aplicações Web e aplicações Desktop
- Porém, o *Model* permanece o mesmo para qualquer tipo de

MVC

- Questões importantes de Design
  - *Model* não deve ter nenhum conhecimento sobre *View* e *Controller*
  - O *Controller* é o centralizador das requisições
  - Pode-se adotar a estratégia de implementar um *Controller* para cada Caso de Uso
    - Existe um padrão de projeto complementar chamado *Front-Controller* (veremos no curso)

Atividade 02

- Prática de laboratório com UML
  - Criar um diagrama de pacotes na ferramenta UML representando a arquitetura que você definiu na atividade anterior (slide 27)
  - Criar um diagrama de classe MVC para um dos casos de uso selecionados na Atividade 01 (visão de caso de uso) da primeira aula.
  - Criar um diagrama de sequências representando um dos fluxos do Caso de Uso