

CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADA – CCESA

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

MÉTODOS AVANÇADOS DE PROGRAMAÇÃO

**Estudo Dirigido**

Um sistema orientado a objetos é composto de objetos que enviam mensagens uns para os outros. O resultado desta troca de informação é a realização de uma operação ou de uma obrigação no sistema. Logo uma obrigação é uma responsabilidade a ser cumprida pelo(s) objeto(s) do sistema.

Uma responsabilidade pode envolver um único método, como criar um item de venda, ou pode envolver dezenas de classes e métodos, como fornecer acesso a um banco de dados. Os métodos são utilizados para implementar responsabilidades.

Assim, definir como distribuir as responsabilidades entre os objetos do sistema é crucial para um bom projeto. Uma má distribuição de responsabilidade conduz a soluções frágeis e difíceis de entender, manter, reusar e estender

O padrão GRASP (General Responsability Assignment Software Patterns, oferece um conjunto de práticas para atribuição de responsabilidades a classes e objetos em projetos orientados a objetos.

- **Dada uma responsabilidade, a qual classe essa responsabilidade deve ser alocada?**

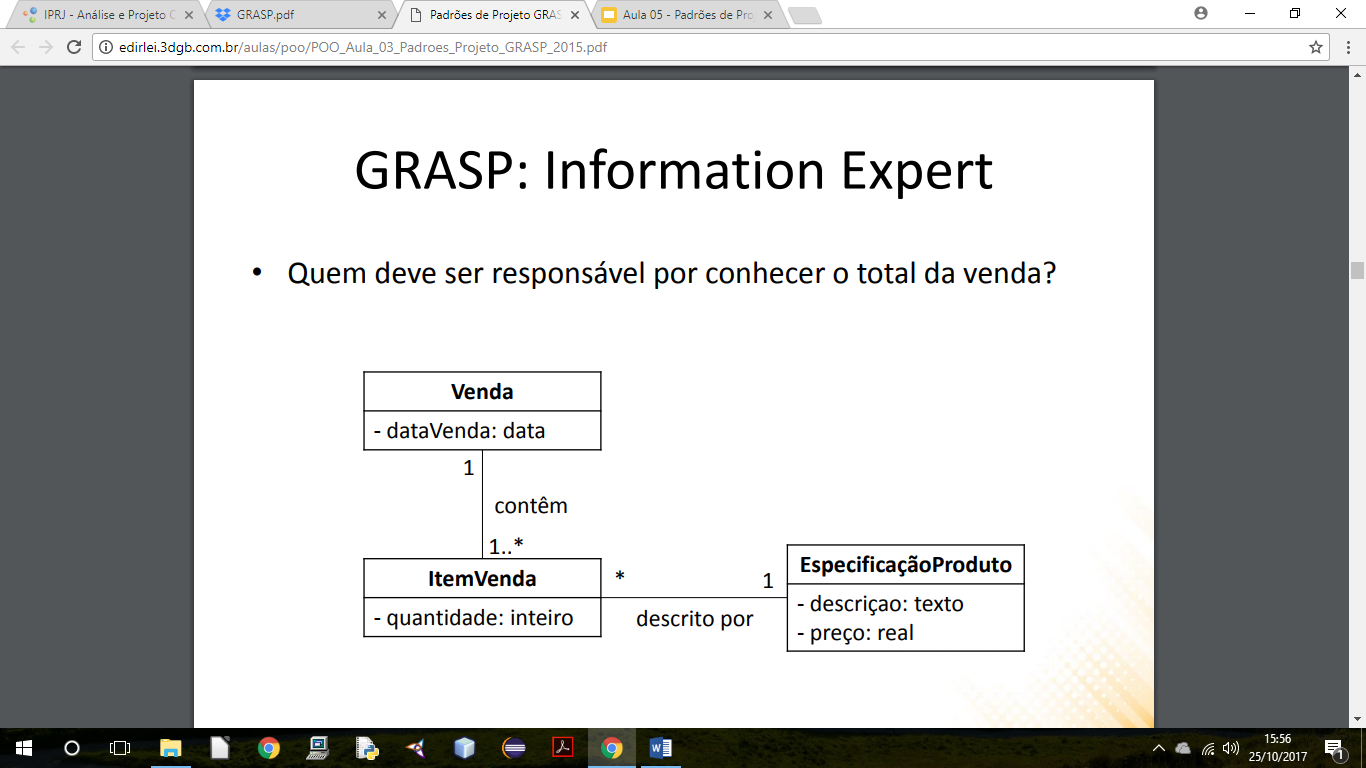
# GRASP

- **Padrões Básicos:** Information Expert; Creator; High Cohesion; Low Coupling; Controller Polimorfismo...

Para a ilustração dos padrões GRASP analisaremos o exemplo de um sistema de vendas e pagamentos. Padrões GRASP são utilizados quase sempre durante a etapa de análise.

Sistema: Grava vendas e gera pagamentos





## Information Expert

* Princípio fundamental para atribuir responsabilidade.
* **Problema**: Qual é o princípio geral para a atribuição de responsabilidades aos objetos?
* **Solução**: Atribua a responsabilidade ao especialista: a classe que tem as informações necessárias para assumir a responsabilidade.

#**Pergunta01**: Quem deve ser o responsável por conhecer o total da venda?

#**Pergunta02**: Quem deve ser o responsável por conhecer o subtotal da venda?

#**Pergunta03**: Quem deve ser o responsável por conhecer o valor de um produto?

Benefícios: Encapsulamento é mantido; Fraco acoplamento (facilidade de manutenção); Alta coesão (objetos fazem tudo relacionado à sua própria informação). No fim, vários especialistas "parciais" podem colaborar.

## Creator

* **Problema**: Quem deve ser responsável por criar uma nova instância de uma classe?
* **Solução**: Atribua à classe B a responsabilidade de criar uma instância de A se pelo menos um desses for verdadeiro (quanto mais melhor):

– B contém ou agrega objetos de A;

– B registra instancias de A;

– B usa muitos objetos de A;

– B tem os dados necessários para a inicialização de A que serão passados ao construtor de A

#**Pergunta04**: Quem deve ser o responsável por criar instâncias do ItemVenda? Precisamos achar alguém que agrega, contém, ... instâncias de ItemVenda.

## Baixo Acoplamento

* Acoplamento é uma medida de quanto um elemento está conectado a, ou depende de outros elementos.

– Uma classe com acoplamento forte depende de muitas outras classes.

– Por isto, acoplamento forte é indesejável.

* **Problema**: Como prover baixa dependência entre classes, reduzir o impacto de mudanças e obter alta reutilização?
* **Solução**: Atribua as responsabilidades de modo que o acoplamento entre classes permaneça baixo. Use este princípio para avaliar alternativas.
* Uma classe com acoplamento forte é:

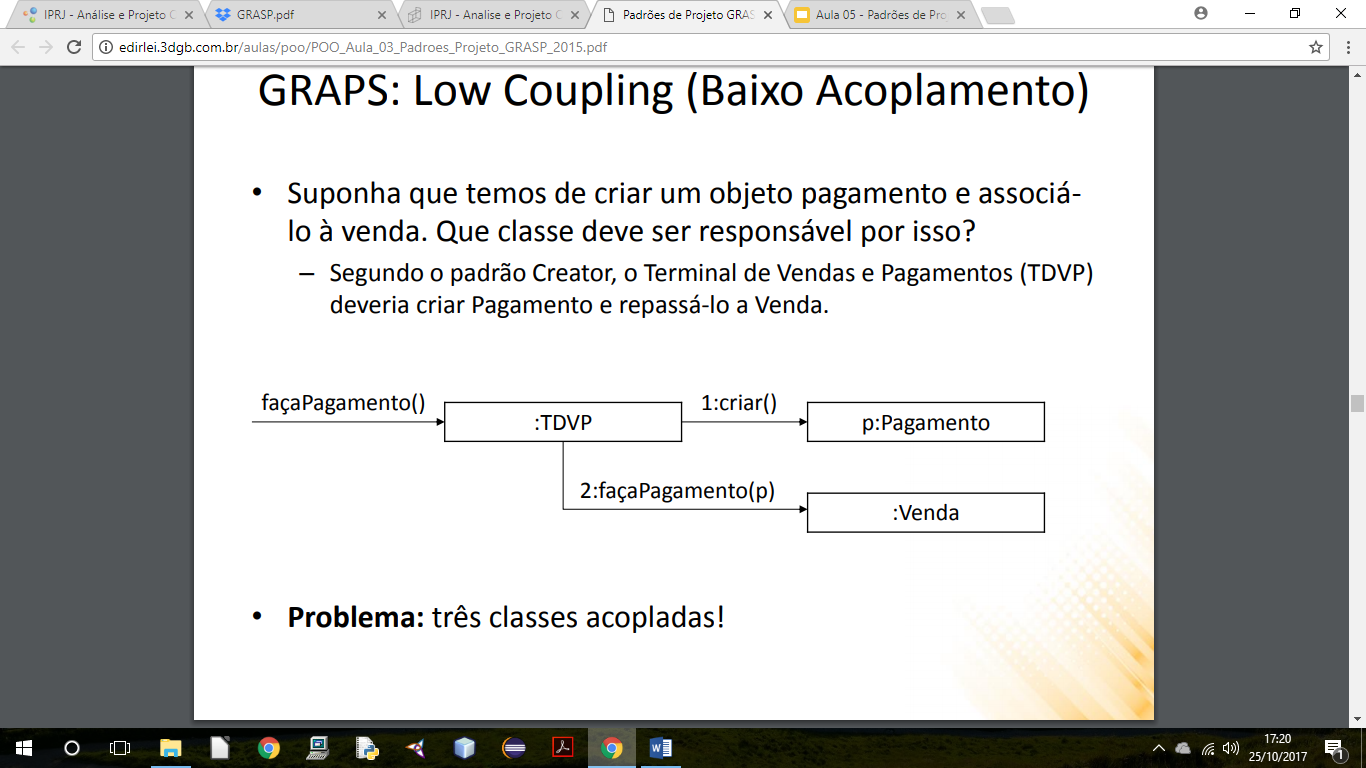
– Mais difícil de compreender isoladamente;

– Mais difícil de reutilizar (seu uso depende da reutilização das outras classes da qual ela depende);

– Sensível a mudanças nas classes associadas.

Analise a seguinte situação: Pretende-se criar um objeto pagamento ao sistema proposto e associá-lo a venda. Observando que existe o terminal de ponto vendas e pagamento (TDVP), o padrão Creator sugere que TDVP crie o Pagamento e repasse para Venda.

Problema! Três classes acopladas:

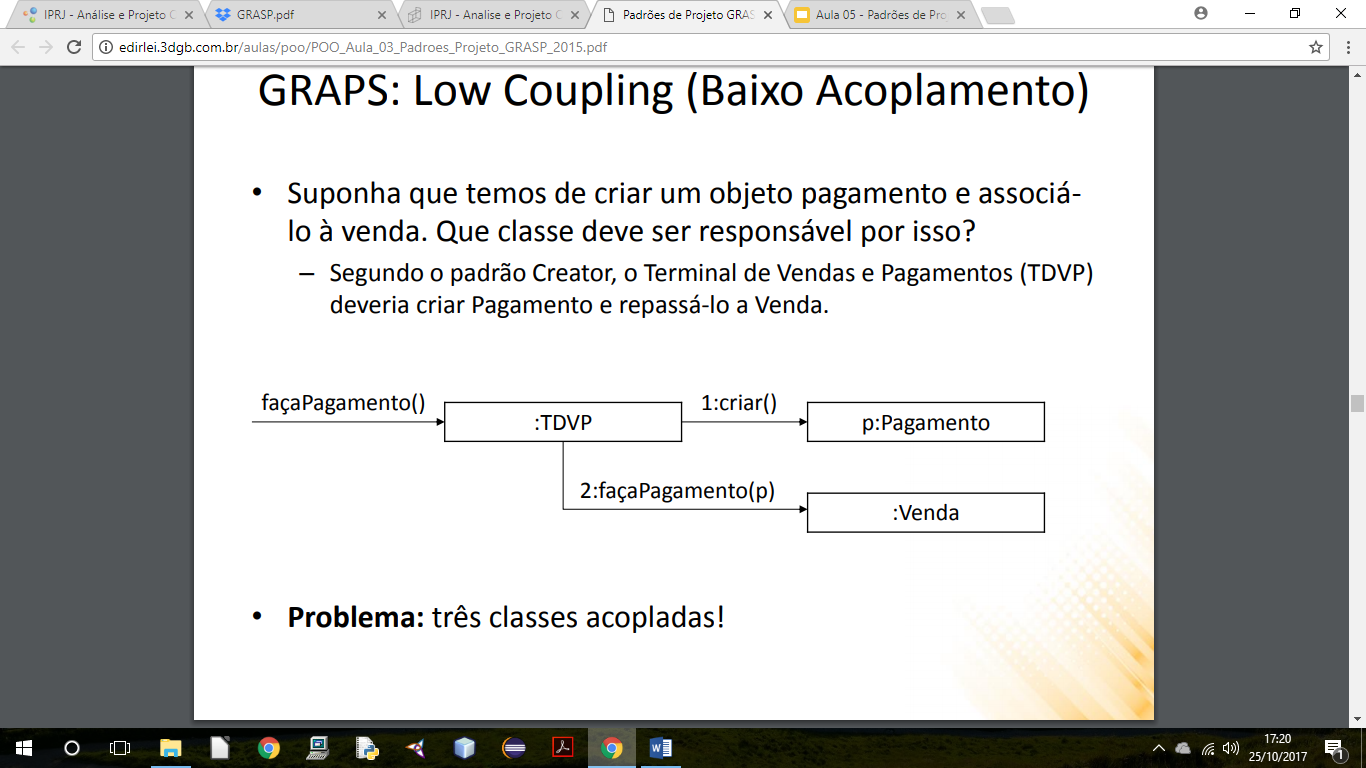


#**Pergunta05**: Como solucionar este problema e reduzir o acoplamento? Tendo em vista que Venda também possui informações necessárias para instanciar o objeto pagamento.

## Alta Coesão

* **Problema**: como manter a complexidade sob controle?
  + Classes que fazem muitas tarefas não relacionadas são:
    - Mais difíceis de entender;
    - Mais difíceis de manter e de reusar;
    - São mais vulneráveis à mudança.
* Solução: atribuir uma responsabilidade para que a coesão se mantenha alta.
* A coesão é uma medida do quão fortemente relacionadas e focalizadas são as responsabilidades de uma classe.
  + Exemplo: uma classe Cachorro é coesa se:
    - Tem operações relacionadas ao Cachorro (morder, correr, comer, latir);
    - E apenas ao Cão (não terá por exemplo listarCaes)

#**Pergunta06**: Que classe de ser responsável por criar um pagamento e associá-lo a uma venda? Mais uma vez seria o TDVP, porém acumularia métodos não relacionados a esta classe.



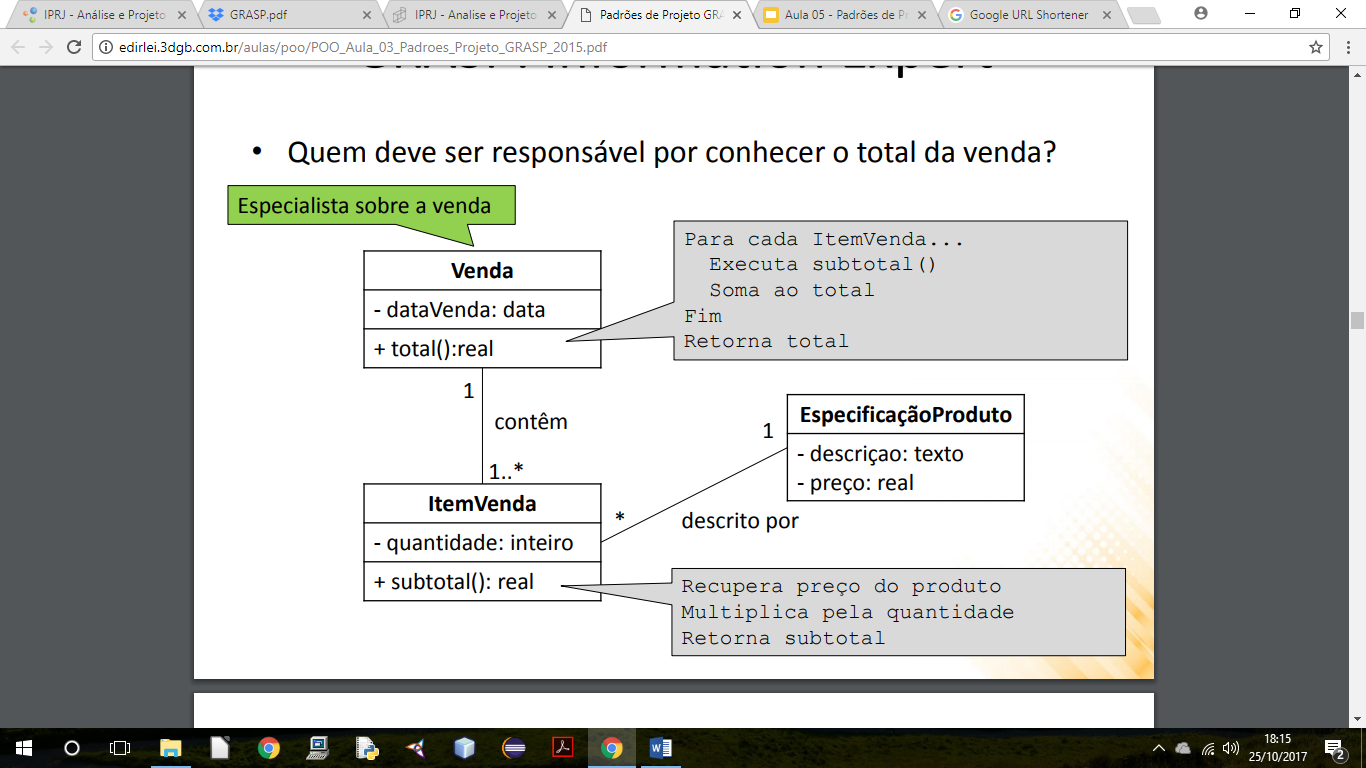
Referências:

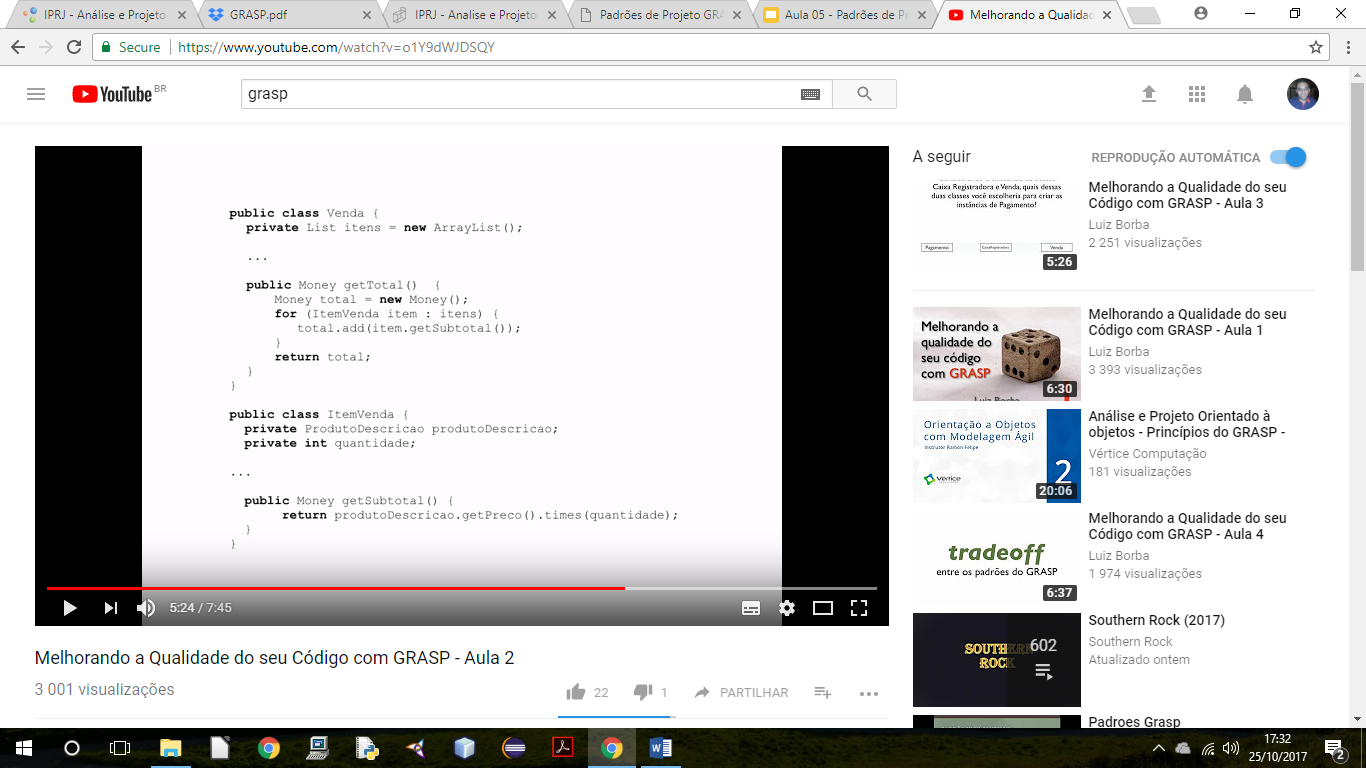
LIMA, E. S. Notas de Aula. 2016

Links úteis: <https://goo.gl/zP5T1X>

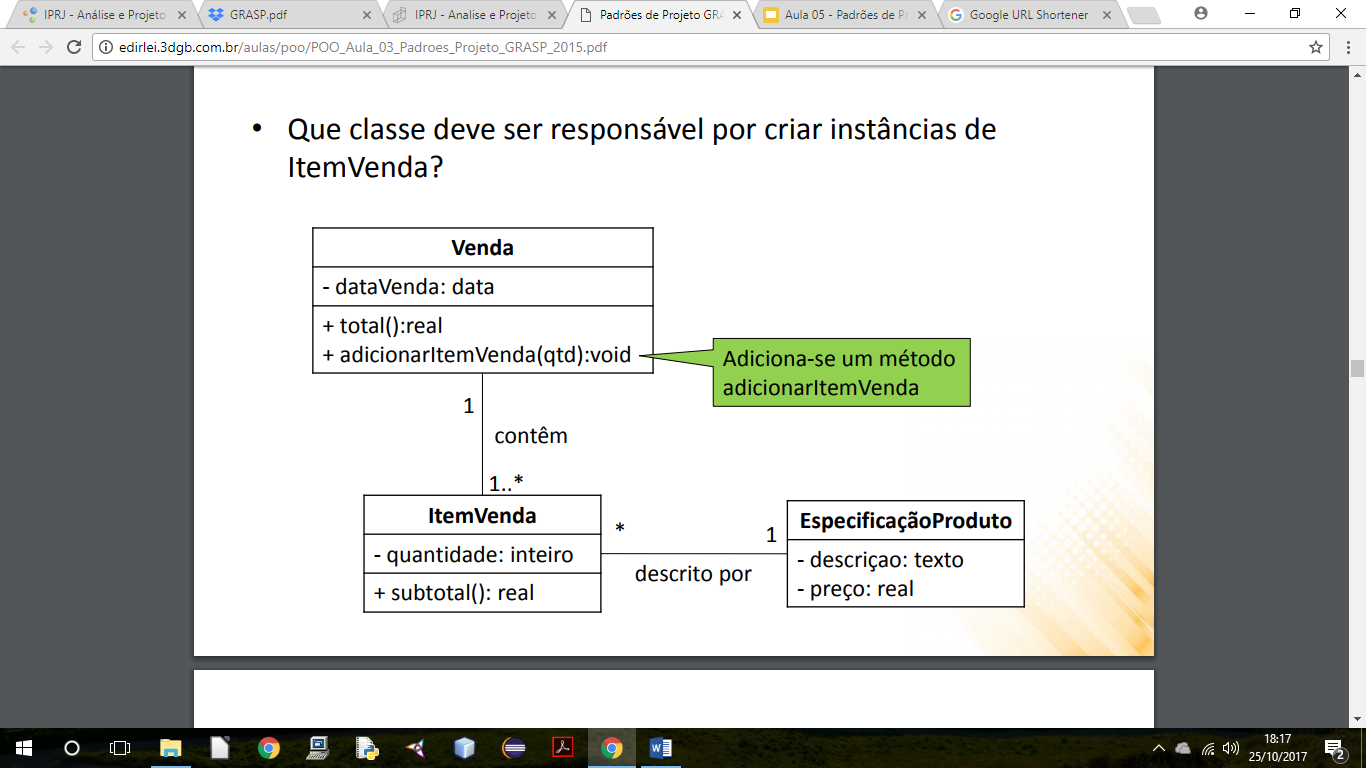
**RESPOSTAS**

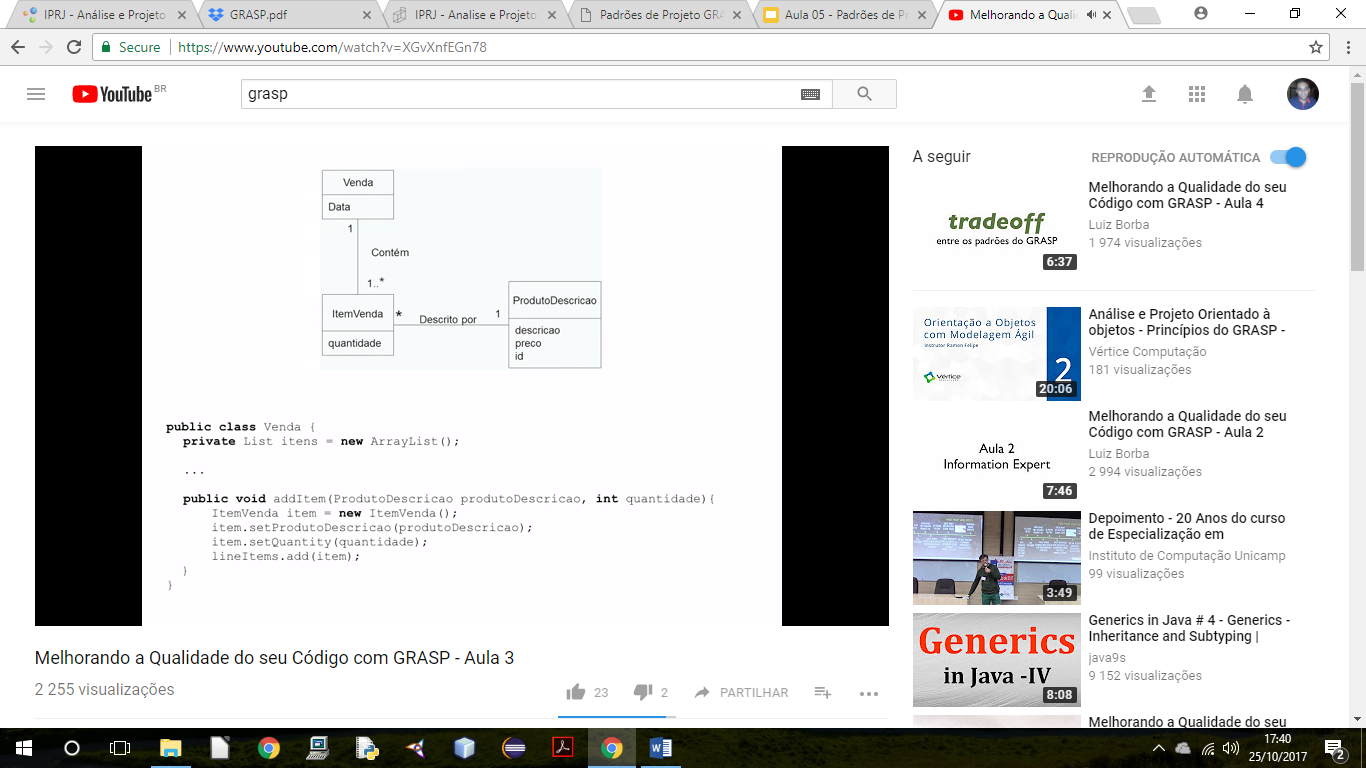
#01 #02 #03



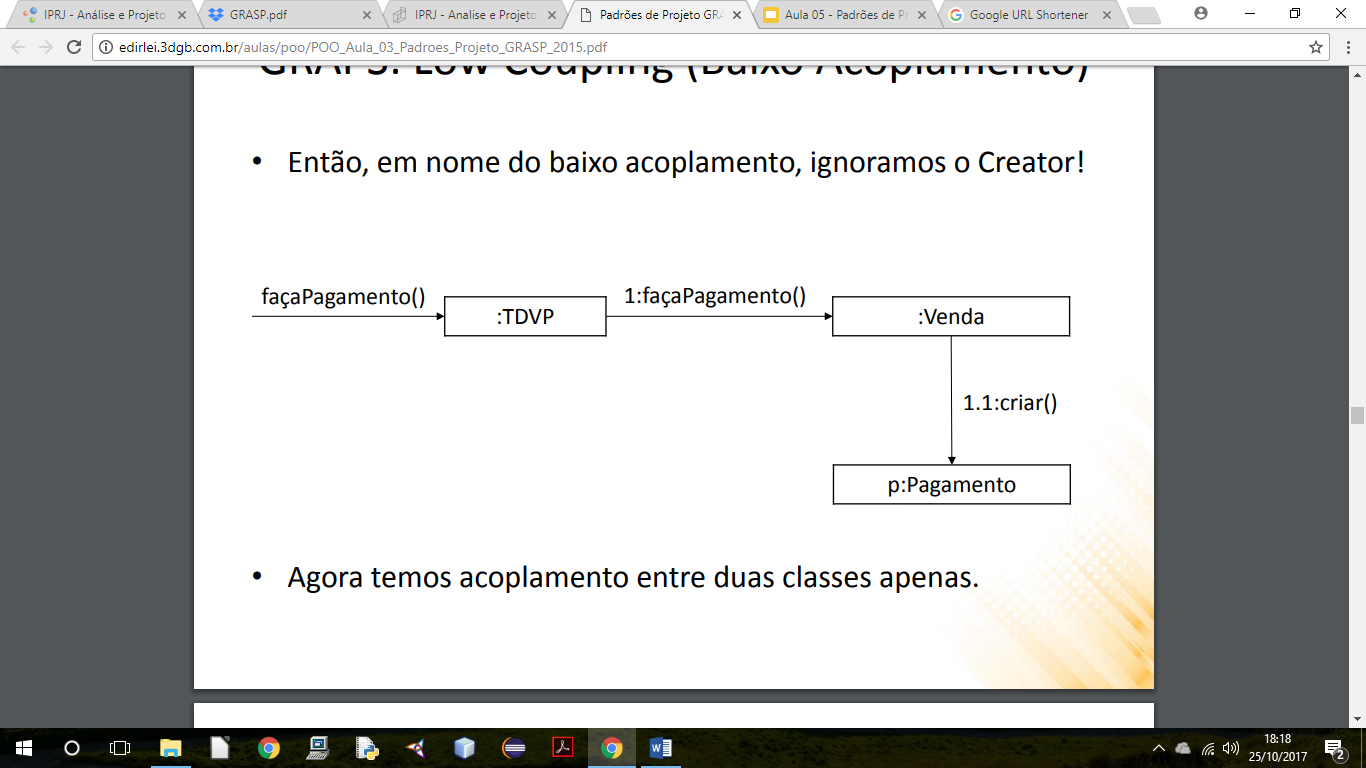


#04

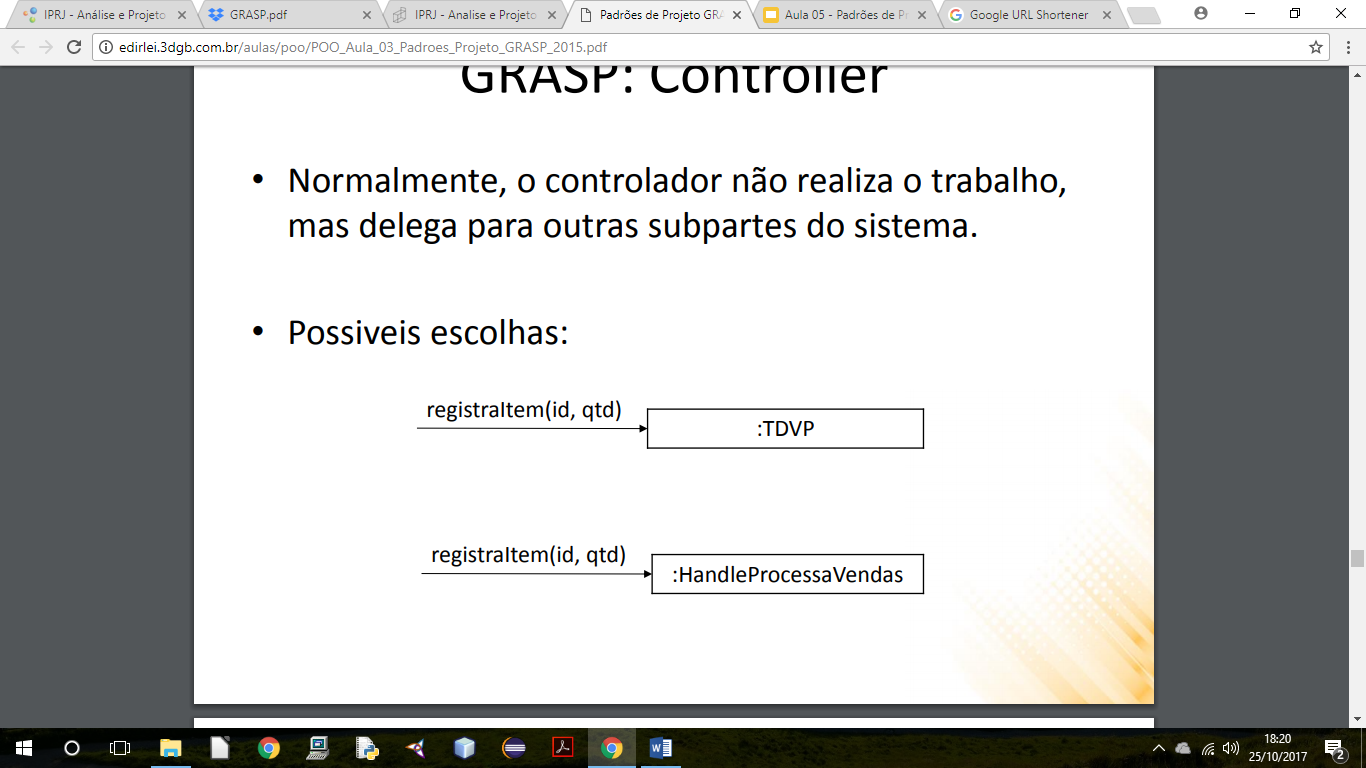




#05 #06



#07



# GoF

#MINITESTE

1. (CESGRANRIO, 2010) Em um sistema de software para controlar pedidos para entrega em domicílio, deve haver uma funcionalidade que permita que o atendente solicite a repetição de um pedido anteriormente feito por um cliente. O gerente do restaurante informou que essa funcionalidade aumentaria a agilidade no atendimento aos clientes, visto que muitos deles tendem a fazer pedidos similares aos que já fizeram anteriormente. Ao usar essa funcionalidade, o atendente do restaurante seleciona um pedido cuja composição corresponde a produtos normalmente requisitados pelos clientes e solicita ao sistema a construção de um novo pedido igual ao selecionado. Esse novo pedido pode, então, ser alterado pelo atendente se o cliente solicitar a adição de novos produtos do cardápio, por exemplo. Portanto, a parte principal dessa funcionalidade corresponde a criar uma cópia de um pedido a partir de pedido preexistente. Na implementação dessa funcionalidade, seu desenvolvedor deve utilizar qual padrão de projeto do catálogo GoF (Gang of Four), dentre os listados abaixo?

a) Builder.

b) Factory Method.

c) Command.

d) Abstract Factory.

e) Prototype.

2. (CESGRANRIO, 2012) O padrão de projeto que tem como finalidade separar a construção de um objeto complexo de sua representação, de forma que um mesmo processo de construção possa criar diferentes representações é conhecido como

a) Abstract Factory

b) Builder

c) Composite

d) Factory Method

e) Prototype

3. (FCC, 2015) Considere:

I. A aplicação cliente interage com uma fábrica genérica de objetos.

II. Os objetos serão gerados efetivamente pela fábrica concreta que estiver sendo utilizada no momento.

III. A aplicação cliente não precisa ser configurada para interagir com cada uma das fábricas concretas.

IV. Novas fábricas concretas podem ser agregadas, alteradas ou retiradas do sistema sem necessidade de alterações na aplicação cliente.

Estas são características e propriedades do padrão de projeto

a) Factory Method.

b) Builder

c) Prototype.

d) Abstract Factory.

e) Singleton.

4. (UFMT, 2015) Qual padrão de projeto visa garantir que um e apenas um objeto seja instanciado para uma dada classe?

a) Factory Method

b) Singleton

c) Abstract Factory

d) Builder

5. (FCC, 2016) Dentre os tipos de Padrões de Projeto (Design Patterns) o que se caracteriza por definir uma interface para a criação de um objeto e que permite que a subclasse decida qual classe instanciar é denominado

a) Factory Method.

b) Builder.

c) Prototype.

d) Abstract Factory.

e) Composite.

**GABARITO**

1. E
2. B
3. D
4. B
5. A