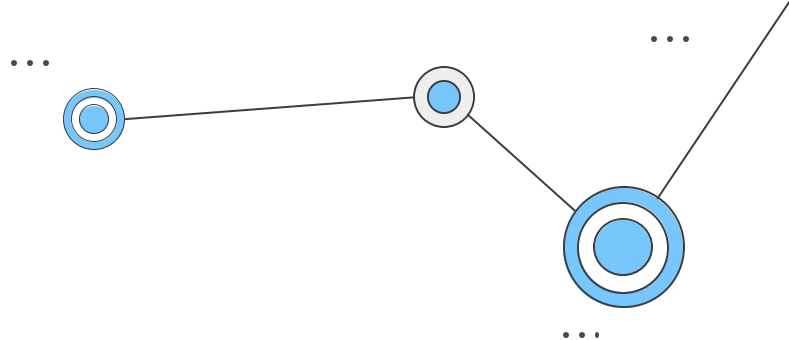


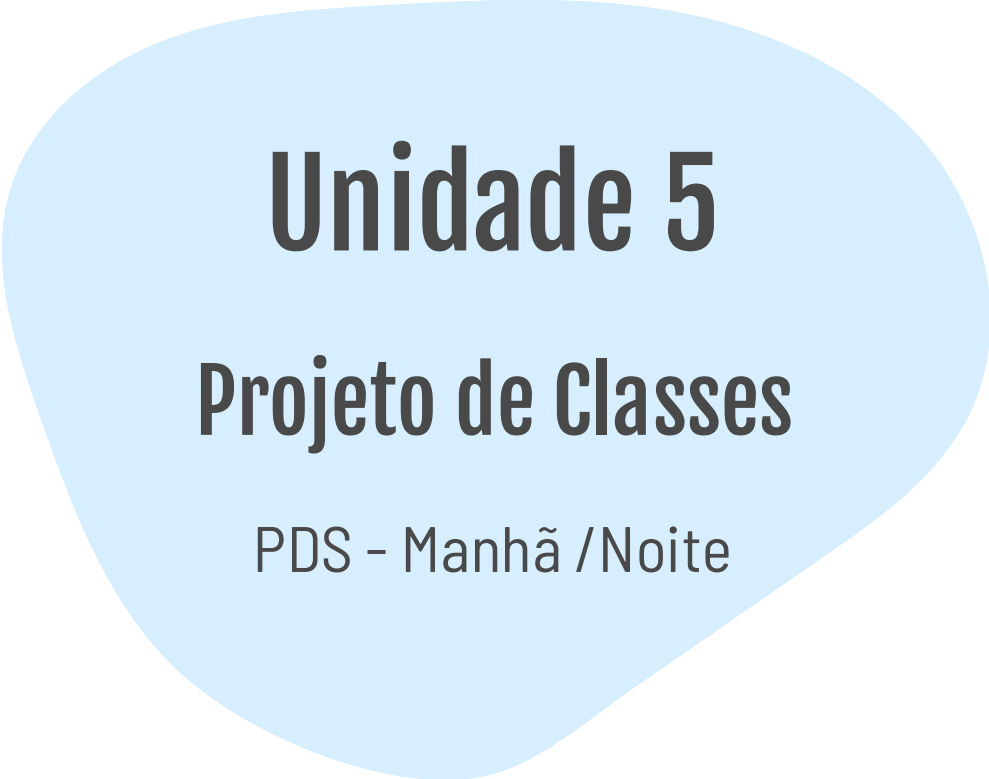
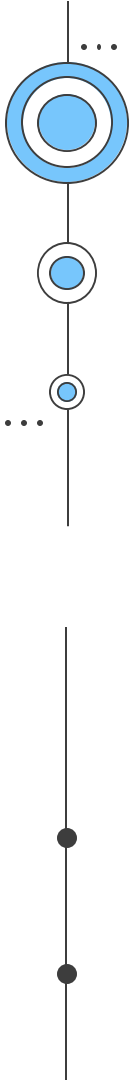


PUC Minas



Projeto de Software

Prof. Dr. João Paulo Aramuni



Unidade 5

Projeto de Classes

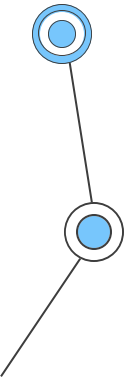
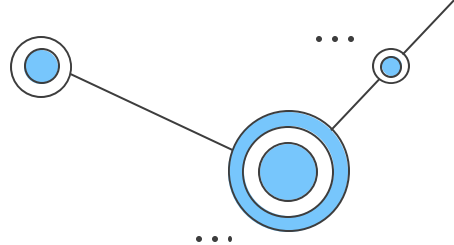
PDS - Manhã /Noite



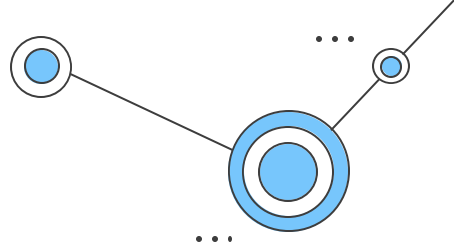
Projeto de Classes

Sumário

- O que é responsabilidade de objetos
- Padrões GRASP
 - Especialista
 - Criador
 - Baixo Acoplamento
 - Alta Coesão
 - Controlador

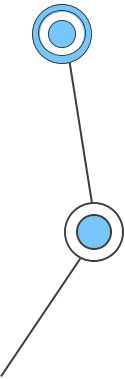


Projeto de Classes

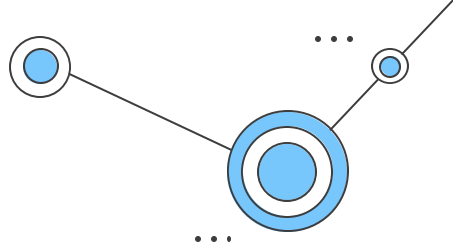


Projeto de Classes

- O que é projeto?
 - É uma das partes mais difíceis da programação.
 - Consiste em criar abstrações.
- Isto significa três coisas:
 - Quais classes devem ser criadas?
 - Quais responsabilidades (operações/métodos) devem ser assumidas por cada classe?
 - Quais são os relacionamentos entre tais classes e objetos dessas classes?
- Criar boas abstrações é difícil e vem com experiência.
 - Porém, algumas regras básicas ajudarão a adquirir a experiência mais rapidamente.



Projeto de Classes



Exemplo de decisões oriundas de projetos
Qual é o problema desse código?

Classe VideoLocadora

fitas : Conjunto;

clienteCorrente : Cliente;

Método emprestaFita(fCodigo: String)

fita : Fita;

emprestimoCorrente : Emprestimo;

item : ItemDeEmprestimo;

fita := fitas.get(fCodigo);

emprestimoCorrente := clienteCorrente.getEmprestimoCorrente();

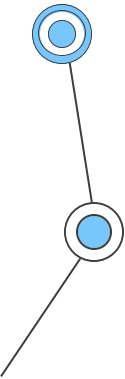
item := ItemDeEmprestimo.new();

item.associaFita(fita);

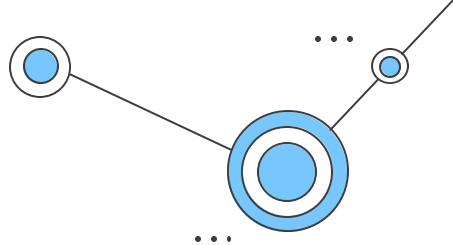
emprestimoCorrente.associaItem(item);

Fim Método;

Fim Classe.



Projeto de Classes



Exemplo de decisões oriundas de projetos

Qual é o problema desse código? **Código com responsabilidades concentradas**

Classe VideoLocadora

fitas : Conjunto;

clienteCorrente : Cliente;

Método emprestaFita(fCodigo: String)

fita : Fita;

emprestimoCorrente : Emprestimo;

item : ItemDeEmprestimo;

fita := fitas.get(fCodigo);

emprestimoCorrente := clienteCorrente.getEmprestimoCorrente();

item := ItemDeEmprestimo.new();

item.associaFita(fita);

emprestimoCorrente.associaItem(item);

Fim Método;

Fim Classe.

Exercício: desenhe o
diagrama de sequencia
para esse código

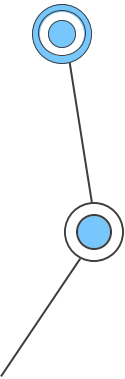
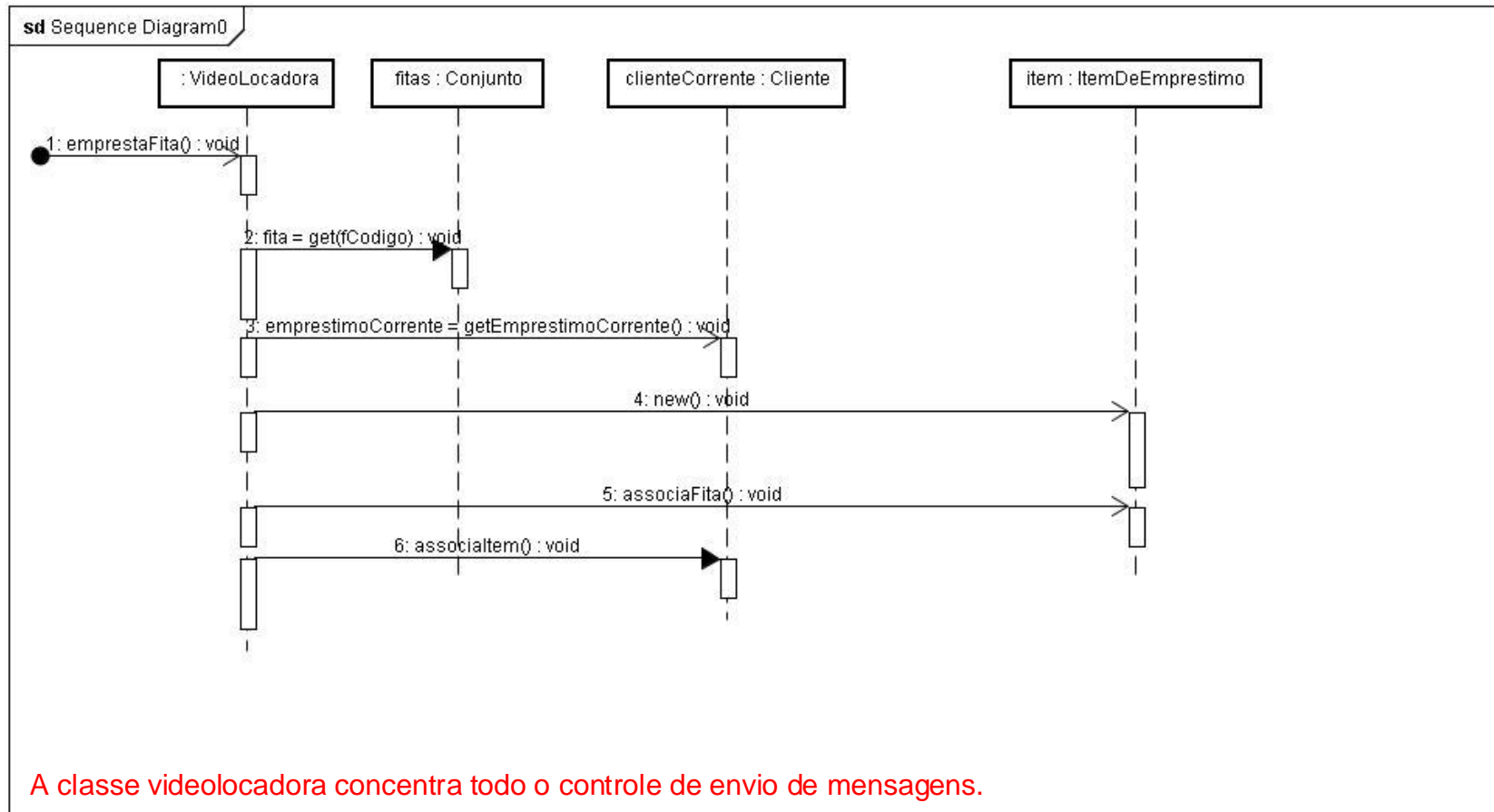


Diagrama de sequência: Código com responsabilidades concentradas



Código com responsabilidades distribuídas

Classe VideoLocadora

fitas : Conjunto ;
clienteCorrente : Cliente;

Metodo emprestaFita(fCodigo : String):

fita : Fita;

fita := fitas.get(fCodigo);
clienteCorrente.empresta(fita)

Fim Metodo;
Fim Classe.

Classe Cliente

emprestimoCorrente : Emprestimo;

Metodo empresta(fita : Fita):

emprestimoCorrente.adiciona(fita);

Fim Metodo;
Fim Classe.

Classe Emprestimo

itens : Conjunto;

Metodo adiciona(fita : Fita):

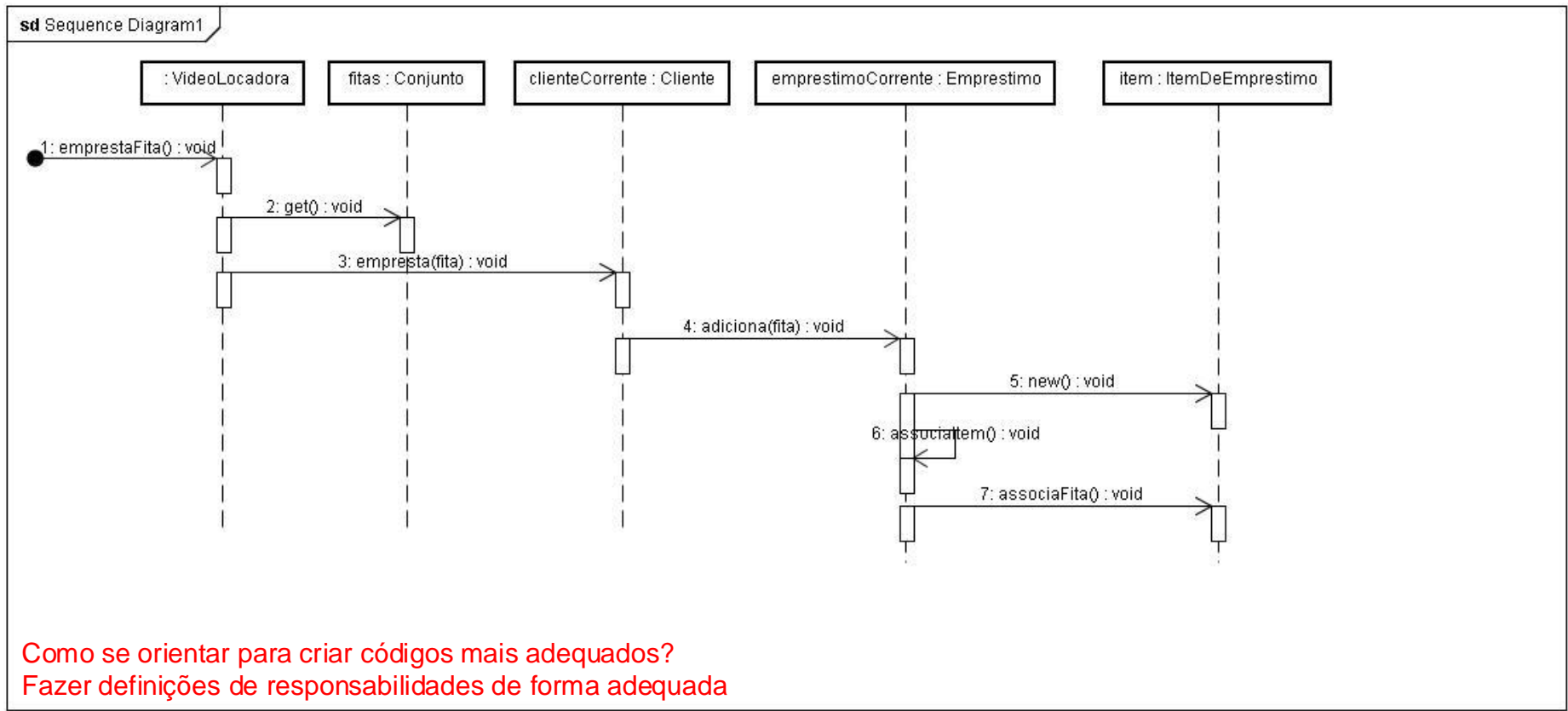
item : ItemDeEmprestimo;

item := ItemDeEmprestimo.new();
self.associaItem(item);
item.associaFita(fita);

Fim Metodo;
Fim Classe.

Exercício: desenhe o
diagrama de sequencia
para esse código

Diagrama de sequência: Código com responsabilidades distribuídas



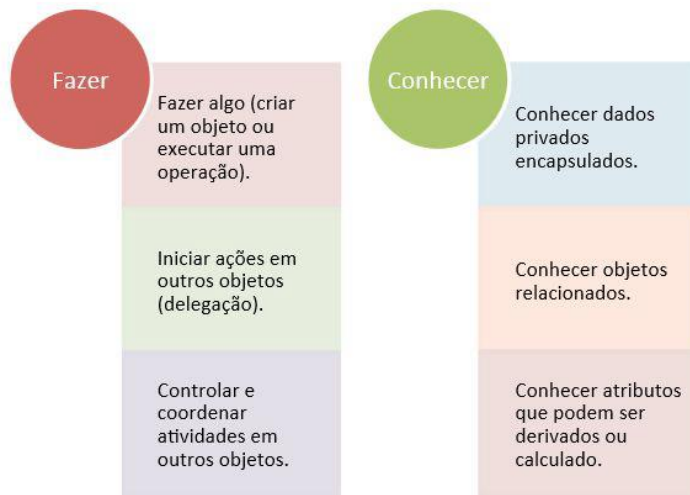
Projeto de Classes

Responsabilidades e Métodos

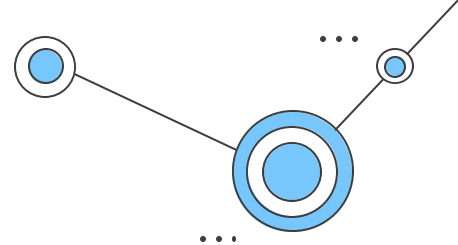
Responsabilidade é “um contrato ou obrigação de um tipo ou classe”
[Booch et al.'97]

- Relacionada com o comportamento dos objetos

Tipos de Responsabilidades



Projeto de Classes



Responsabilidades e Métodos

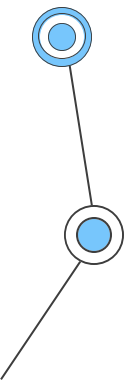
Responsabilidades são atribuídas aos objetos do sistema durante o projeto OO .

Exemplos:

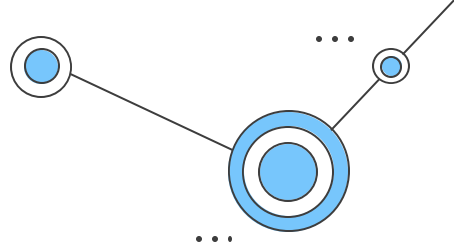
- ✓ “Uma Venda é responsável por imprimir a si própria” (de fazer)
- ✓ “Uma Venda é responsável por conhecer sua data” (de conhecer)
- ✓ Uma Conta bancária tem a responsabilidade de logar as transações (de fazer)
- ✓ Uma Conta bancária tem a responsabilidade de saber sua data de criação (de conhecer)

Tradução para classes e métodos é influenciada pela granularidade da responsabilidade

- Um único método para “imprimir venda”
- Dezenas de métodos para “prover um mecanismo de acesso a SGBD relacionais”



Projeto de Classes



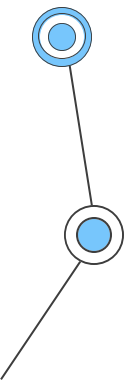
Responsabilidades e Métodos

Métodos são implementados para cumprir responsabilidades.

- Podem agir sozinhos ou em colaboração com outros métodos e objetos.

Exemplo:

- A classe Venda pode definir um método específico para cumprir a responsabilidade de impressão.
- Esse método, por sua vez, pode precisar colaborar com outros objetos, possivelmente enviando mensagens de impressão para cada um dos objetos Item-de-Venda associados à Venda.

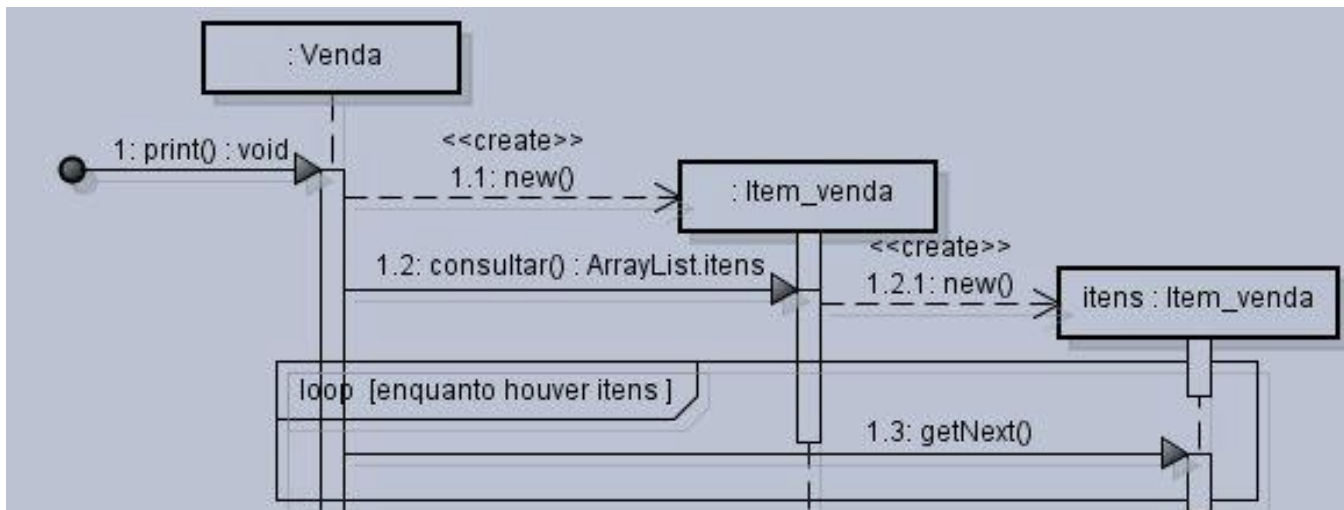


Projeto de Classes

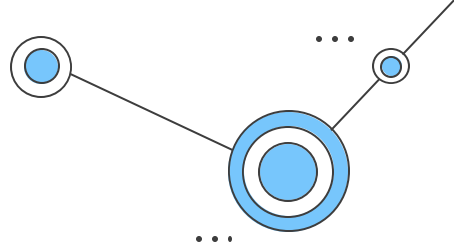
Responsabilidades e Interações

Diagramas de interação ilustram decisões na atribuição de responsabilidades a objetos.

- Quais mensagens são enviadas para diferentes classes e objetos?



Projeto de Classes

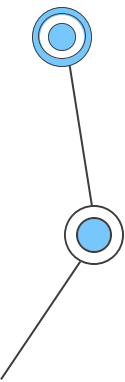


Padrões de Projetos

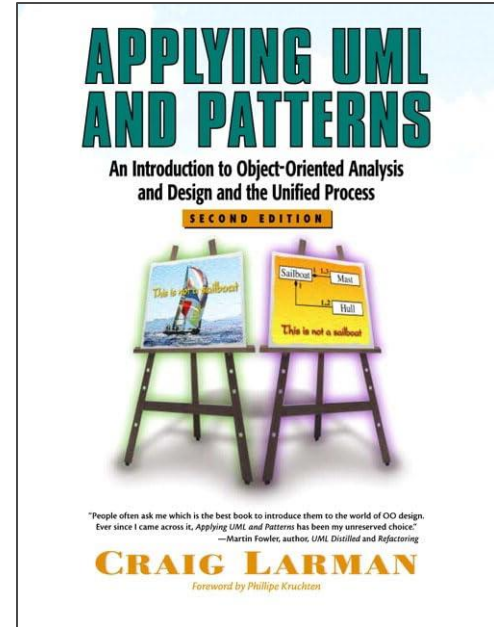
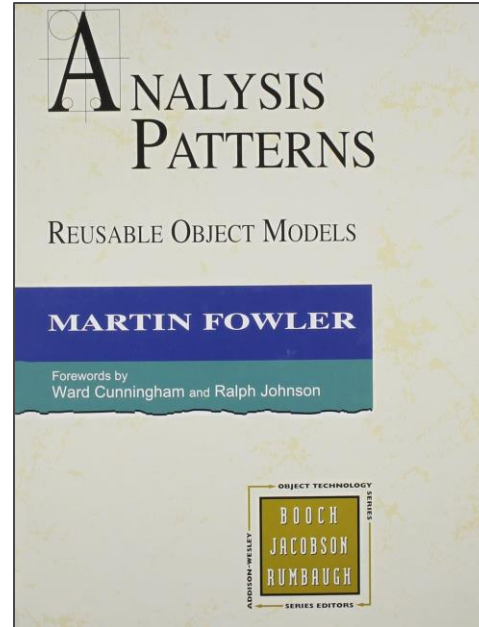
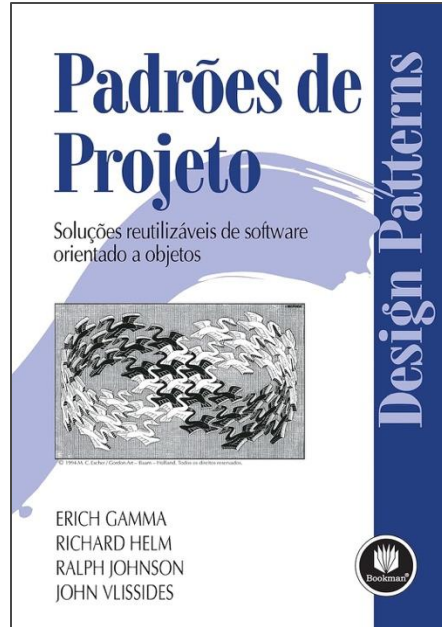
Uma dúvida recorrente para projetistas OO/UML é sobre a alocação de métodos em classes.

- Quais métodos criar?
- Que classes devo ter?
- Quais classes devem conter quais métodos?

Padrões de projetos são utilizados para uniformizar estratégias para criação de estruturas orientadas a objetos, em nível de relacionamento entre classes e em nível de interação por mensagens.



Projeto de Classes



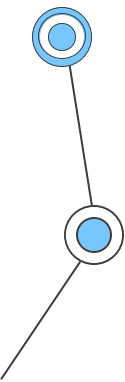
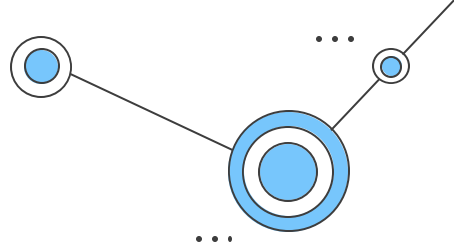
Projeto de Classes

Padrões de Projetos

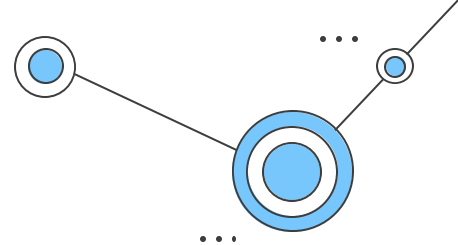
Expressam uma solução para um determinado problema em um determinado contexto incluindo:

- Nome (facilita a abstração e comunicação).
- Problema que ele resolve.
- Solução para o problema.
- Conselhos sobre sua aplicação em novas situações.
- Discussão sobre consequências de seu uso.

Criados por desenvolvedores experientes.

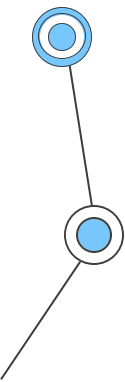


Projeto de Classes



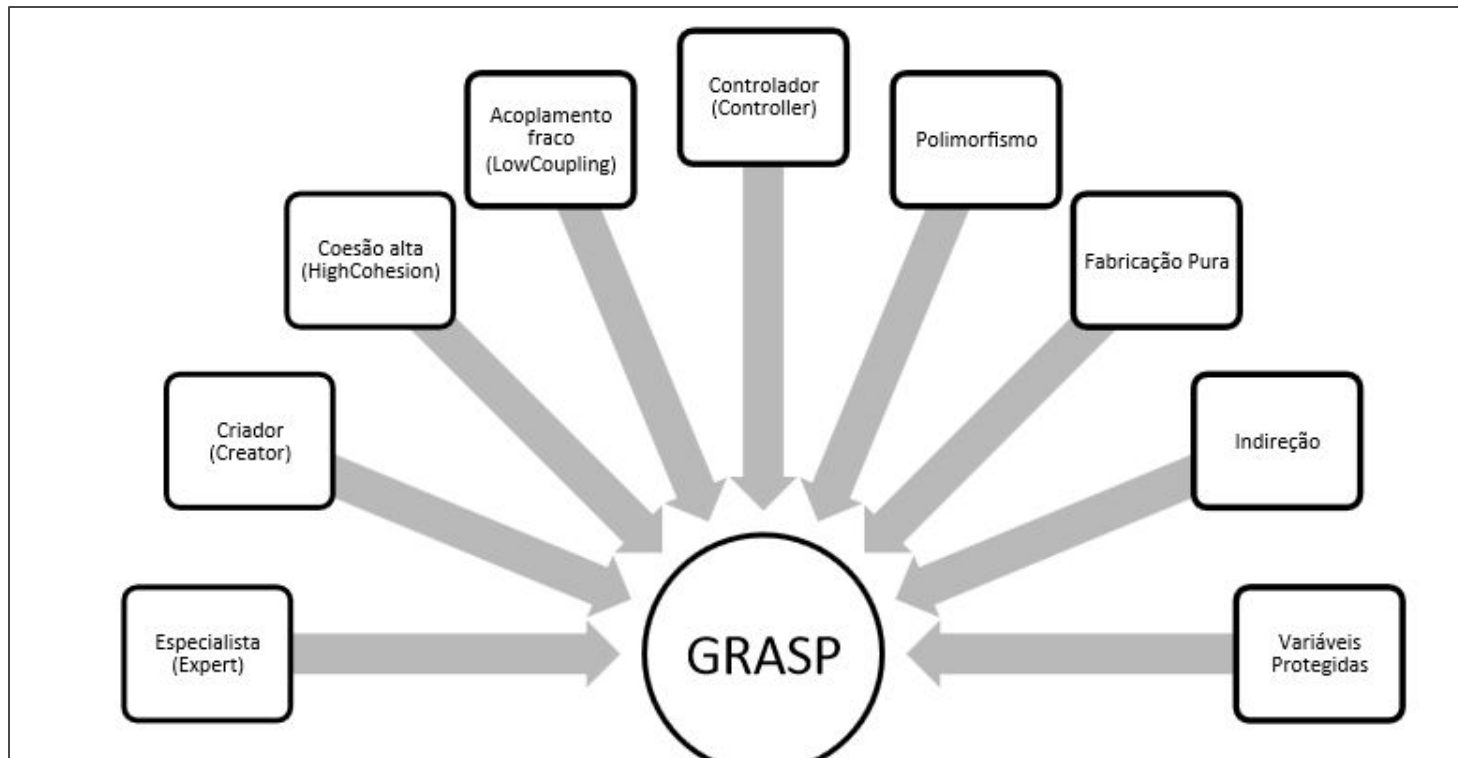
Padrões GRASP: General Responsibility Assignment Software Patterns

- Define nove princípios básicos de projeto OO ou blocos construtivos básicos de projetos.
 - Cinco padrões são considerados fundamentais.
 - Quatro são considerados padrões 'avançados'.
- O que os padrões GRASP fazem?
 - Os padrões GRASP descrevem princípios fundamentais de projeto baseado em objetos e atribuição de responsabilidades aos mesmos.
- Por que os padrões GRASP são importantes?
 - Um desenvolvedor novato na tecnologia de objetos necessita dominar os princípios básicos rapidamente.
 - Padrões GRASP são a base de um projeto de sistema.



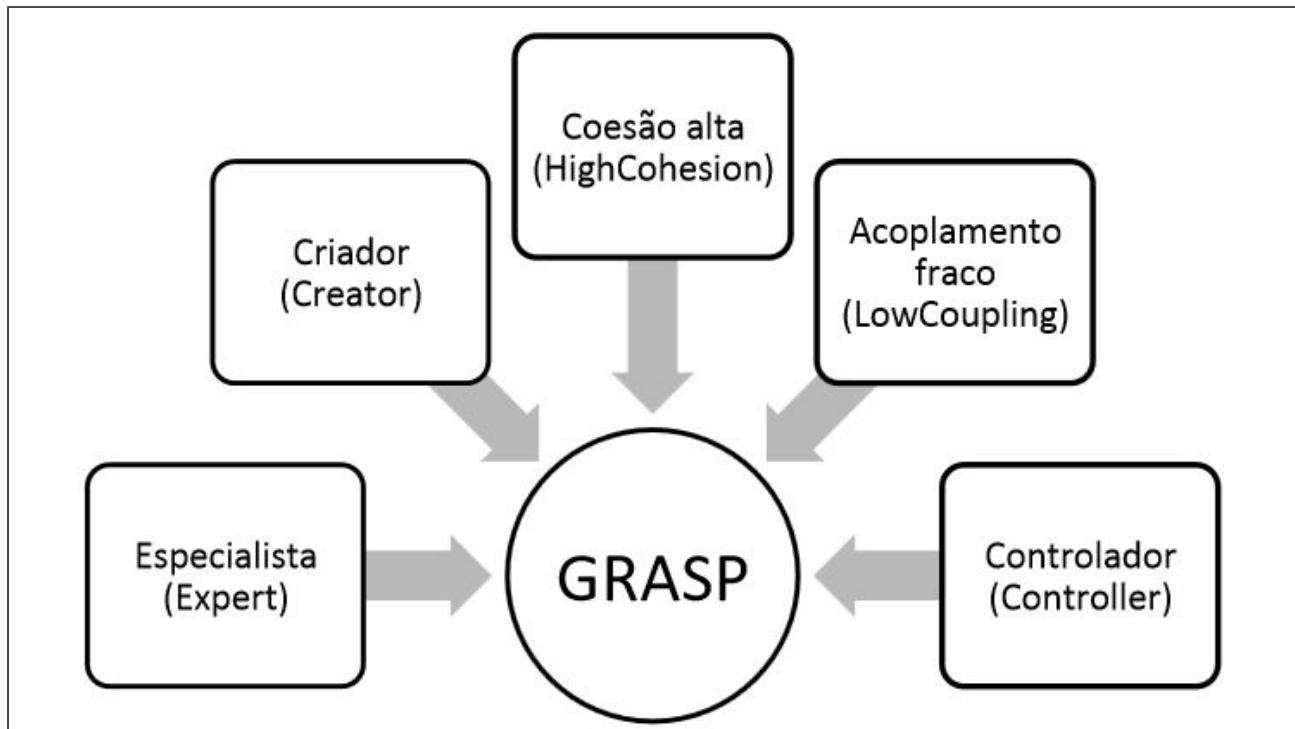
Projeto de Classes

Padrões GRASP: General Responsibility Assignment Software Patterns



Projeto de Classes

Padrões GRASP: Fundamentais

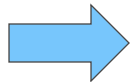


Projeto de Classes

Padrões GRASP

Padrões básicos

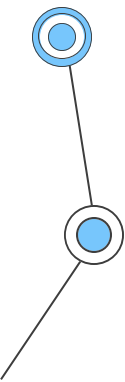
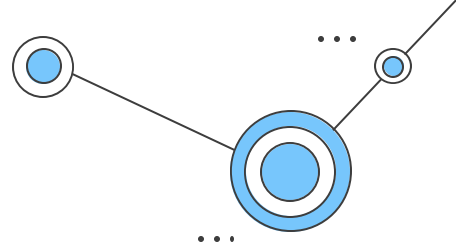
- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller



Padrões avançados

- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

- Padrões GRASP refletem práticas mais pontuais da aplicação de técnicas OO
- Padrões GoF exploram soluções mais específicas.
- Padrões GRASP ocorrem na implementação de vários padrões GoF.

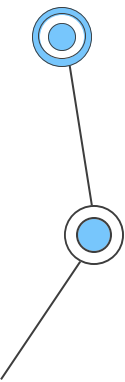
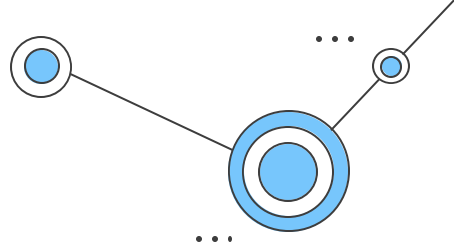


Projeto de Classes

"GoF" é a abreviação de "Gang of Four", que se refere aos quatro autores do livro clássico "***Design Patterns: Elements of Reusable Object-Oriented Software***". Os autores são Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides.

Este livro, publicado em 1994, é amplamente reconhecido como uma obra seminal no campo do design de software orientado a objetos.

Os Padrões GoF fornecem soluções específicas e práticas para problemas recorrentes no design de software orientado a objetos, enquanto os padrões GRASP focam na atribuição geral de responsabilidades. Ambos os conjuntos de padrões são complementares e essenciais para um design de software eficaz e robusto.



Projeto de Classes

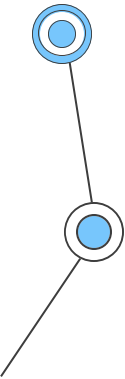
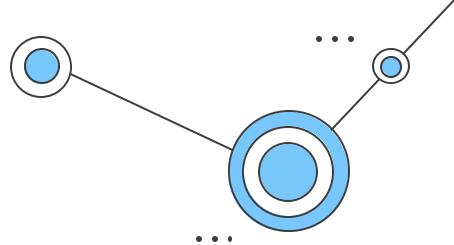
Padrão Especialista da Informação

Problema:

- Qual é o princípio mais básico pelo qual responsabilidades são atribuídas no POO?

Solução:

- Atribuir responsabilidade para o especialista na informação
 - ✓ a classe que tem a informação necessária para cumprir a responsabilidade.

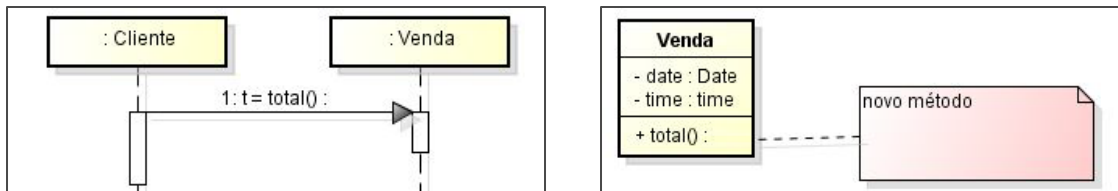


Projeto de Classes

Padrão Especialista

Exemplo:

- Quem deve ser responsável por conhecer o total de uma venda?
 - ✓ Que informação é necessária para determinar o total geral? conhecer todas as instâncias Item_Venda da Venda e o subtotal de cada uma delas.
- Pelo padrão, a classe Venda deve ser a responsável, pois ela conhece a informação e é a especialista na informação.



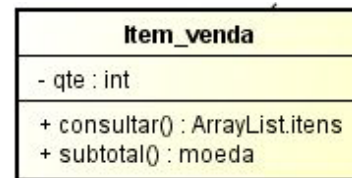
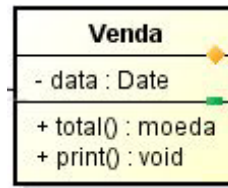
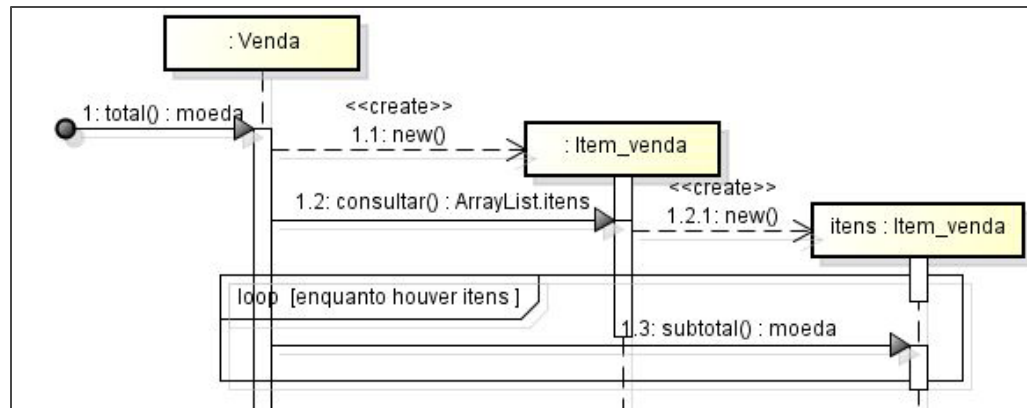
- Mas quem dever ser responsável por conhecer o subtotal de um Item-de-Venda?
 - ✓ Informação necessária:
 - Item_Venda.quantidade
 - Produto.preço

Projeto de Classes

Padrão Especialista

Exemplo (cont.)

- Pelo padrão, a classe Item_Venda deve ser a responsável.



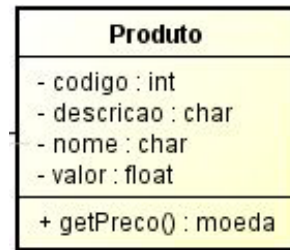
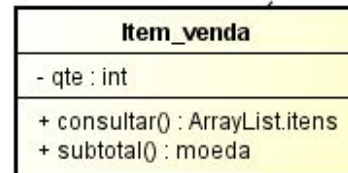
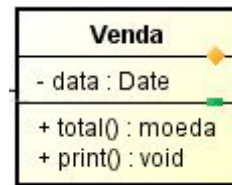
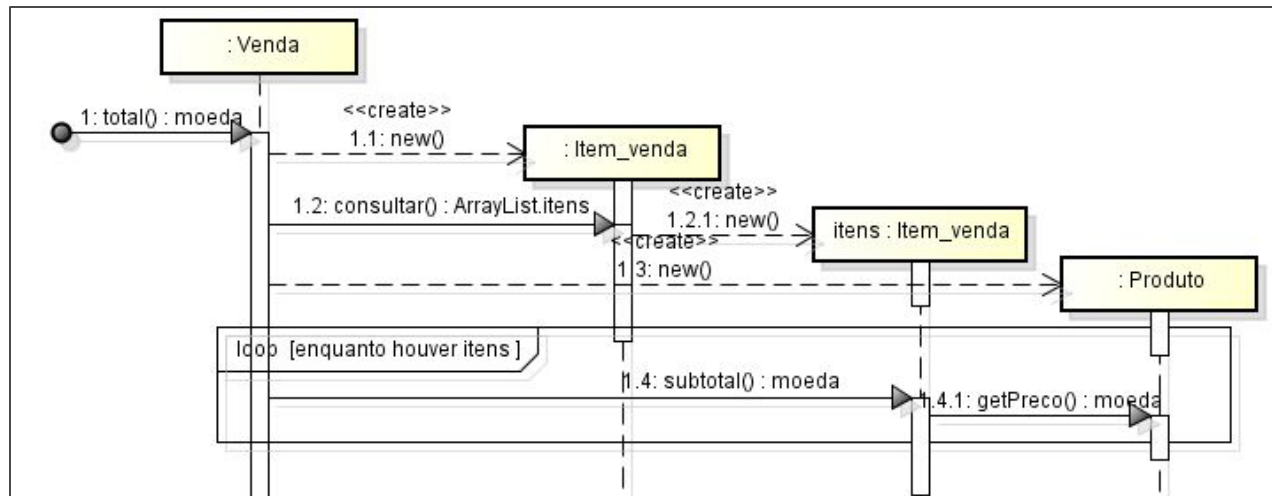
- Porém, para cumprir essa responsabilidade um Item_Venda precisa conhecer o preço do Item.

Projeto de Classes

Padrão Especialista

Exemplo (cont.)

- Portanto, o Item_Venda deve mandar uma mensagem para a Produto para saber o preço do item.



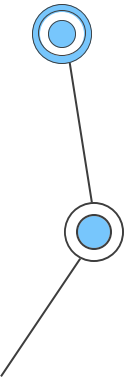
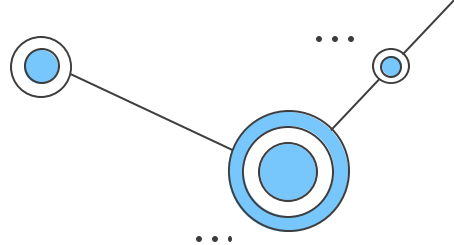
Projeto de Classes

Padrão Especialista

Exemplo (cont.)

- Concluindo, para satisfazer a responsabilidade de conhecer e comunicar o total da venda, três responsabilidades foram atribuídas a três classes de objetos:

Classe	Responsabilidades
Venda	conhece total da venda
Item_Venda	conhece subtotal do item
Produto	Conhece preço do produto



Projeto de Classes

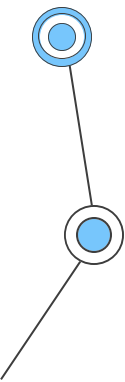
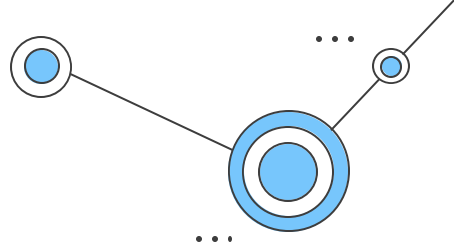
Padrão Especialista

Benefícios:

- Mantém encapsulamento (baixo acoplamento).
- Comportamento é distribuído através das classes que tem a informação necessária para cumprir a responsabilidade (alta coesão).

Contraindicações:

- Viola a separação dos principais interesses.
 - ✓ Por exemplo: lógica e controle.



Projeto de Classes

Pseudocódigo que **não** atende ao padrão Especialista

Classe Supermercado

clienteCorrente : *Cliente*;

consulta valorTotalDaVendaCorrente():

venda : *Venda*;

item : *ItemDeVenda*;

total : *Moeda* = 0,00;

venda := *clienteCorrente*.getVendaCorrente();

repita para cada item em *venda*.getItensDeVenda():

total := *total* + (*item*.getQuantidade() * *item*.getProduto().getPreco());

fim repita

retorna *total*

fim consulta

Fim Classe.

Exercício: desenhe o diagrama de sequência para esse código

Projeto de Classes

Pseudocódigo que atende ao padrão Especialista

Classe Supermercado

clienteCorrente : Cliente;

consulta valorTotalDaVendaCorrente();

retorna clienteCorrente.getValorTotalDaVendaCorrente();

fim consulta

Fim Classe.

Classe Cliente

vendaCorrente : Venda;

consulta getValorTotalDaVendaCorrente();

retorna vendaCorrente.getValorTotal();

fim consulta

Fim Classe

Classe Venda

itens : Conjunto de ItemDeVenda;

total : Moeda = 0,00;

consulta getValorTotal();

repita para cada item em itens:

total := total + item.getSubtotal();

fim repita

retorna total;

fim consulta

Fim Classe

Classe ItemDeVenda

produto : Produto;

consulta getSubtotal();

retorna getQuantidade() * produto.getPreco();

fim consulta

Fim Classe

Exercício: desenhe o diagrama de sequencia para esse código

Projeto de Classes

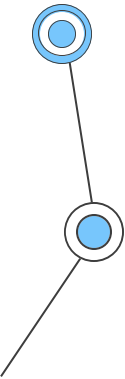
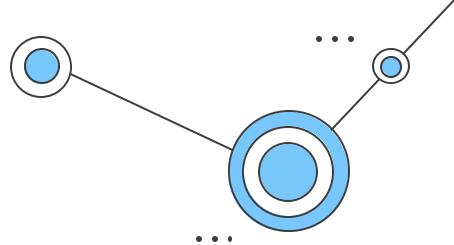
Padrão Criador

Problema:

- Quem deve ser responsável por criar uma nova instância de alguma classe?

Solução:

- Atribuir à classe B a responsabilidade de criar uma instância da classe A se uma das alternativas abaixo for verdadeira:
 - ✓ B agrega ou contém instâncias de A
 - ✓ B registra instâncias de A
 - ✓ B usa instâncias A de maneira muito próxima
 - ✓ B tem os dados de inicialização para criar instâncias de A B portanto é um especialista na criação de A

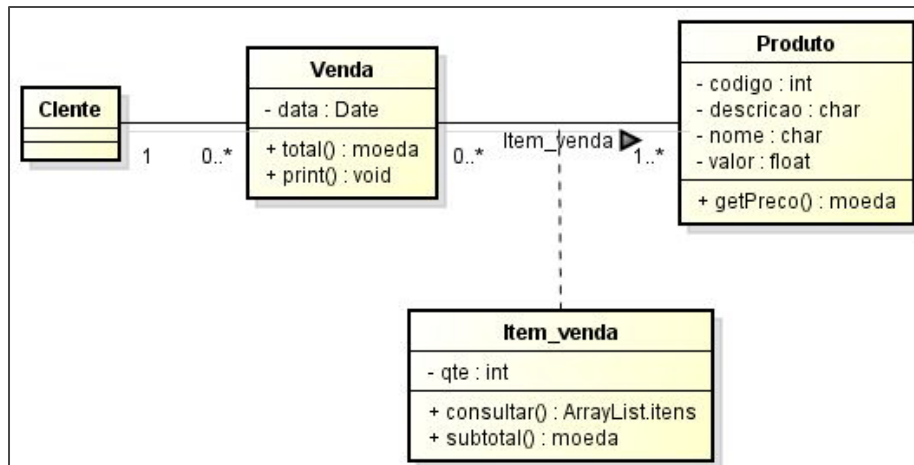


Projeto de Classes

Padrão Criador

Exemplo:

- Quem deve ser responsável por criar uma instância de Item_Venda?
- Pelo padrão Creator, precisamos achar alguém que agrega, contém, ... instâncias de Item_Venda.
- Considere o modelo conceitual parcial abaixo:

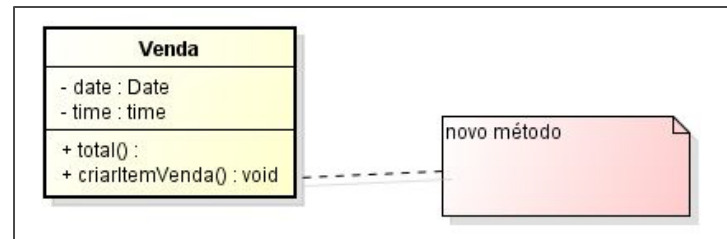
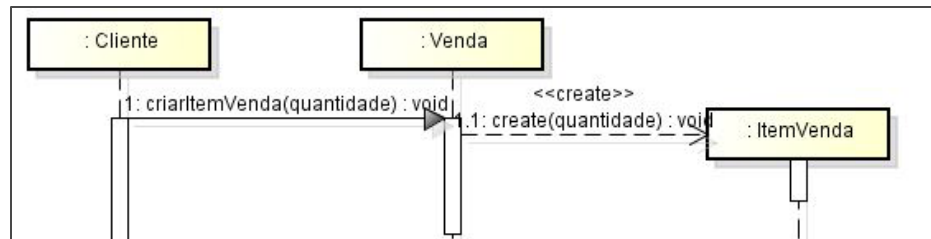


Projeto de Classes

Padrão Criador

Exemplo:

- Venda agrega instâncias de Item_Venda e é portanto um bom candidato para criar as instâncias
- Pelo padrão, Venda deve ser responsável, pois contém ou agrega muitos objetos de Item_Venda.
- Chegamos aos seguintes diagramas:



```
Public class Venda { ...
Public void criarItemVenda(int qte) {....
    item=new ItemVenda(qte)...} }
```


Projeto de Classes

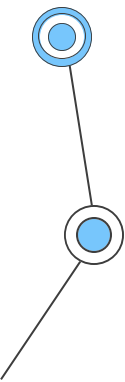
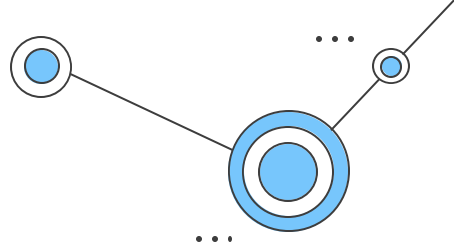
Padrão Criador

Contraindicação:

- Não é indicado se a criação de um objeto for uma tarefa complexa.
- Delegar a criação a uma classe auxiliar chamada Fábrica.

Benefícios:

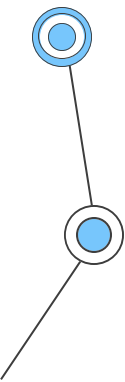
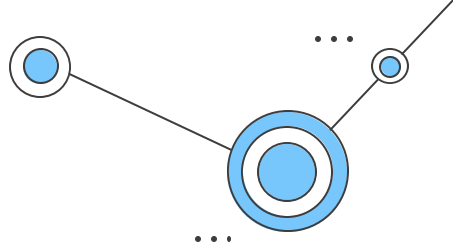
- Acoplamento fraco.
 - ✓ não aumenta o acoplamento pois provavelmente a classe criada já é visível à classe criadora devido às associações.



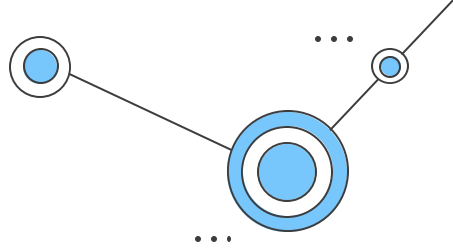
Projeto de Classes

Padrão Criador

- Escolhemos um criador que estará conectado ao objeto criado, de qualquer forma, depois da criação.
- Isso leva a fraco acoplamento.
- Exemplo de criador que possui os valores de inicialização.
 - Uma instância de Pagamento deve ser criada.
 - A instância deve receber o total da venda.
 - Quem tem essa informação? Venda.
 - Venda é um bom candidato para criar objetos da classe Pagamento.



Projeto de Classes



Padrão Acoplamento Fraco

Problema:

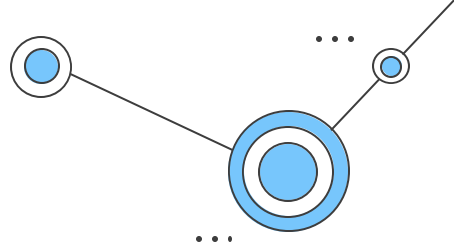
- Como conseguir menor dependência (entre as classes), o pequeno impacto à mudanças e aumentar a reutilização?
- O acoplamento é uma medida de quão fortemente uma classe está conectada, possui conhecimento ou depende de outra classe.
- Com fraco acoplamento, uma classe não é dependente de muitas outras classes.
- Com uma classe possuindo forte acoplamento, temos os seguintes problemas:
 - ✓ Mudanças em uma classe relacionada força mudanças locais à classe.
 - ✓ A classe é mais difícil de entender isoladamente.
 - ✓ A classe é mais difícil de ser reusada, já que depende da presença de outras classes.

Solução:

- Atribuir a responsabilidade de modo que o acoplamento (dependência entre classes) permaneça baixo.



Projeto de Classes

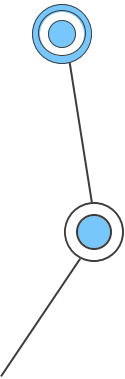


Padrão Acoplamento Fraco – Acoplamento OO

Como o acoplamento se manifesta:

- O TipoX tem um atributo (membro de dados ou variável de instância) que referencia uma instância do TipoY ou o próprio TipoY.
- Um objeto do TipoX chama os serviços de um objeto do TipoY.
- O TipoX tem um método que referencia uma instância do TipoY, ou o próprio TipoY, de alguma forma.
 - ✓ Isso normalmente inclui um parâmetro ou variável local de TipoY, ou
 - ✓ então o objeto retornado por uma mensagem pode ser uma instância do TipoY.
- O TipoX é uma subclasse direta ou indireta do TipoY.
- O TipoY é uma interface e o TipoX implementa essa interface.

Não se deve minimizar acoplamento criando alguns poucos objetos monstruosos (God classes).

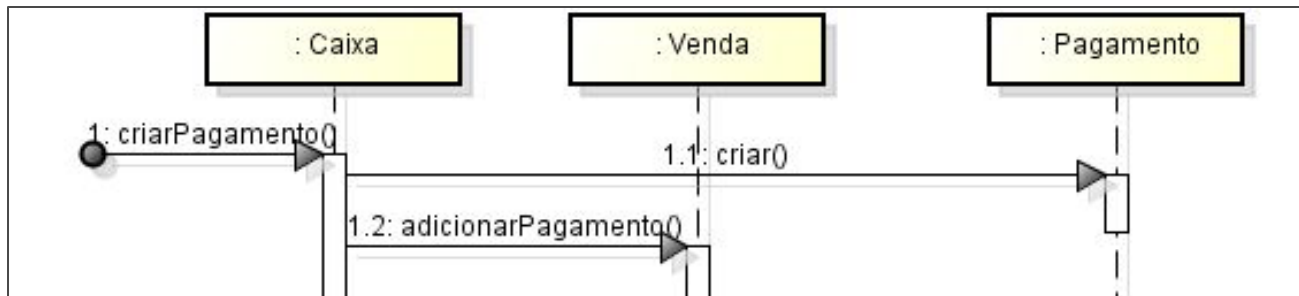


Projeto de Classes

Padrão Baixo Acoplamento

Exemplo

- Quem deve ser responsável por criar um Pagamento e associá-lo à Venda?
- Pelo padrão Criador, poderia ser Caixa (uma vez que Caixa “registra” pagamentos no mundo real).

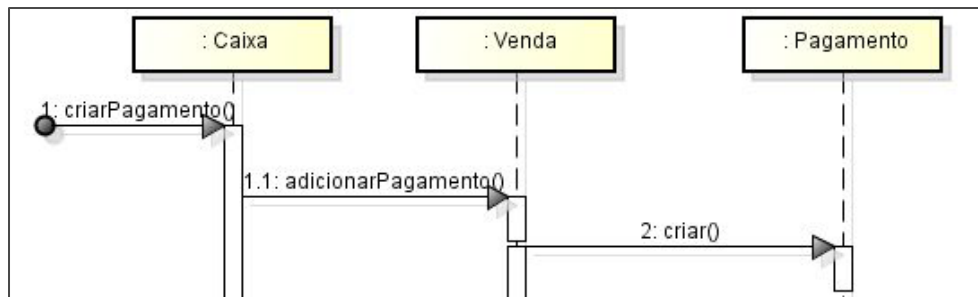


Projeto de Classes

Padrão Baixo Acoplamento

Exemplo (cont.)

- Para reduzir o acoplamento, Venda deve criar o Pagamento:



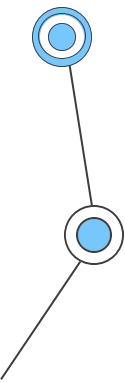
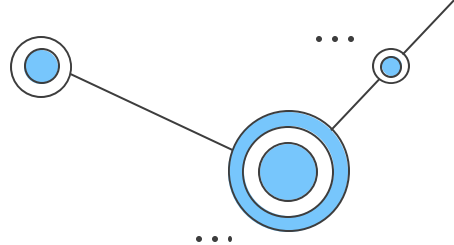
- Supondo que a Venda deva ter conhecimento do pagamento (depois da criação) de qualquer jeito, essa alternativa tem menos acoplamento (Caixa não está acoplado a Pagamento).
- Dois padrões (Creator e Low Coupling) sugeriram diferentes soluções. Nesse caso, minimizar acoplamento foi priorizado.

Projeto de Classes

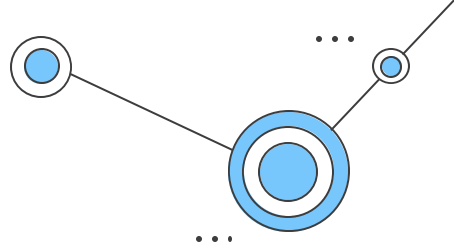
Padrão Baixo Acoplamento

Benefícios:

- Responsabilidade de uma classe não é (ou é pouco) afetada por mudanças em outros componentes.
- Responsabilidade de uma classe é mais simples de entender isoladamente.
- Aumenta a chance de reutilização de uma classe.



Projeto de Classes



Padrão Alta coesão

Problema:

- Como manter a complexidade (das classes) em um nível “controlável”? Isto é, Como gerenciar a complexidade?
- A coesão mede quão relacionadas ou focadas estão as responsabilidades da classe.
- Uma classe com baixa coesão faz muitas coisas não relacionadas e leva aos seguintes problemas:
 - ✓ Difícil de entender
 - ✓ Difícil de reusar
 - ✓ Difícil de manter
 - ✓ "Delicada": constantemente sendo afetada por outras mudanças.
- Uma classe com baixa coesão assumiu responsabilidades que pertencem a outras classes.

Solução:

- Atribuir a responsabilidade de modo que a coesão (força do relacionamento entre as responsabilidades de uma classe) permaneça alta.

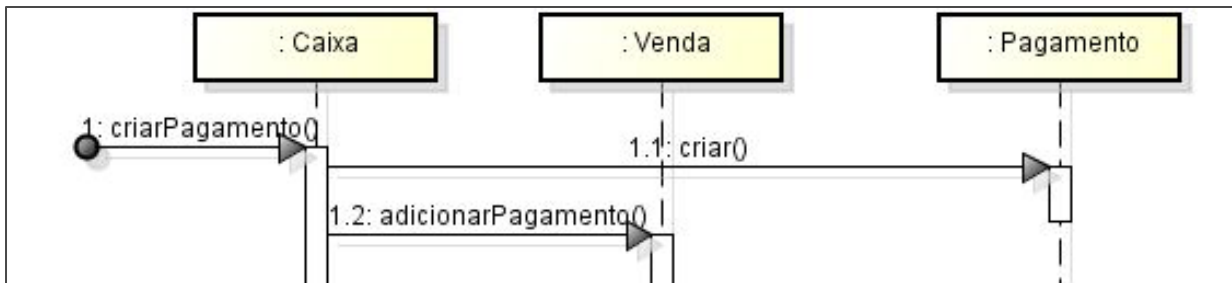


Projeto de Classes

Padrão Alta coesão

Exemplo:

- Mesmo exemplo utilizado no padrão baixo acoplamento.
- Quem deve ser responsável por criar um Pagamento e associá-lo à Venda?
- Pelo padrão Criador, seria Caixa.



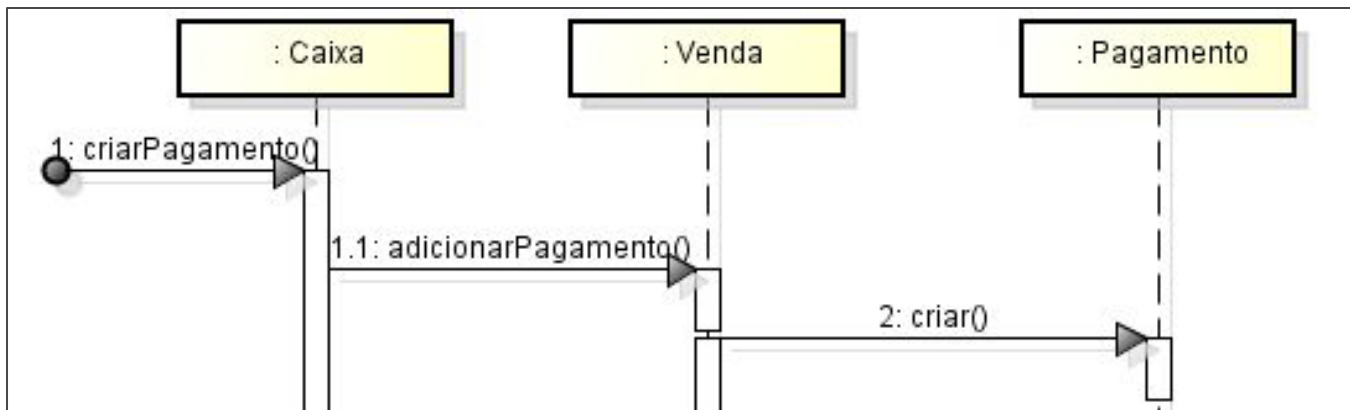
- Mas se Caixa for o responsável pela maioria das operações do sistema, ele vai ficar cada vez mais sobrecarregado:
 - ✓ Resultado é baixa coesão.

Projeto de Classes

Padrão Alta coesão

Exemplo (cont.)

- A criação de Pagamento deve ser delegada a Venda para favorecer uma coesão alta e um acoplamento baixo.

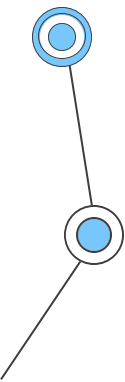
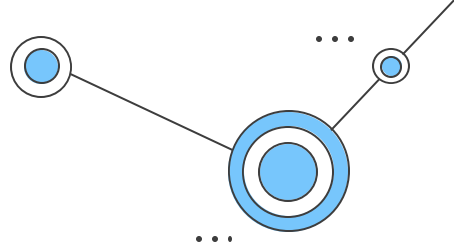


Projeto de Classes

Padrão Alta Coesão

Benefícios:

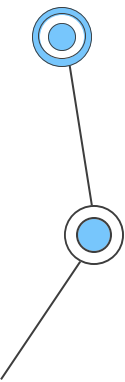
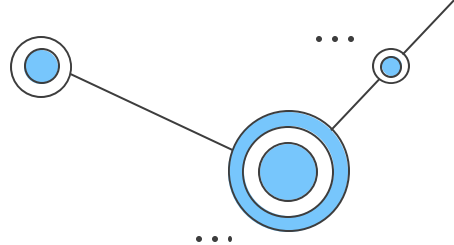
- Aumento da clareza e compreensão do projeto.
- Simplificação de manutenção.
- Baixo acoplamento.
- Reuso facilitado.



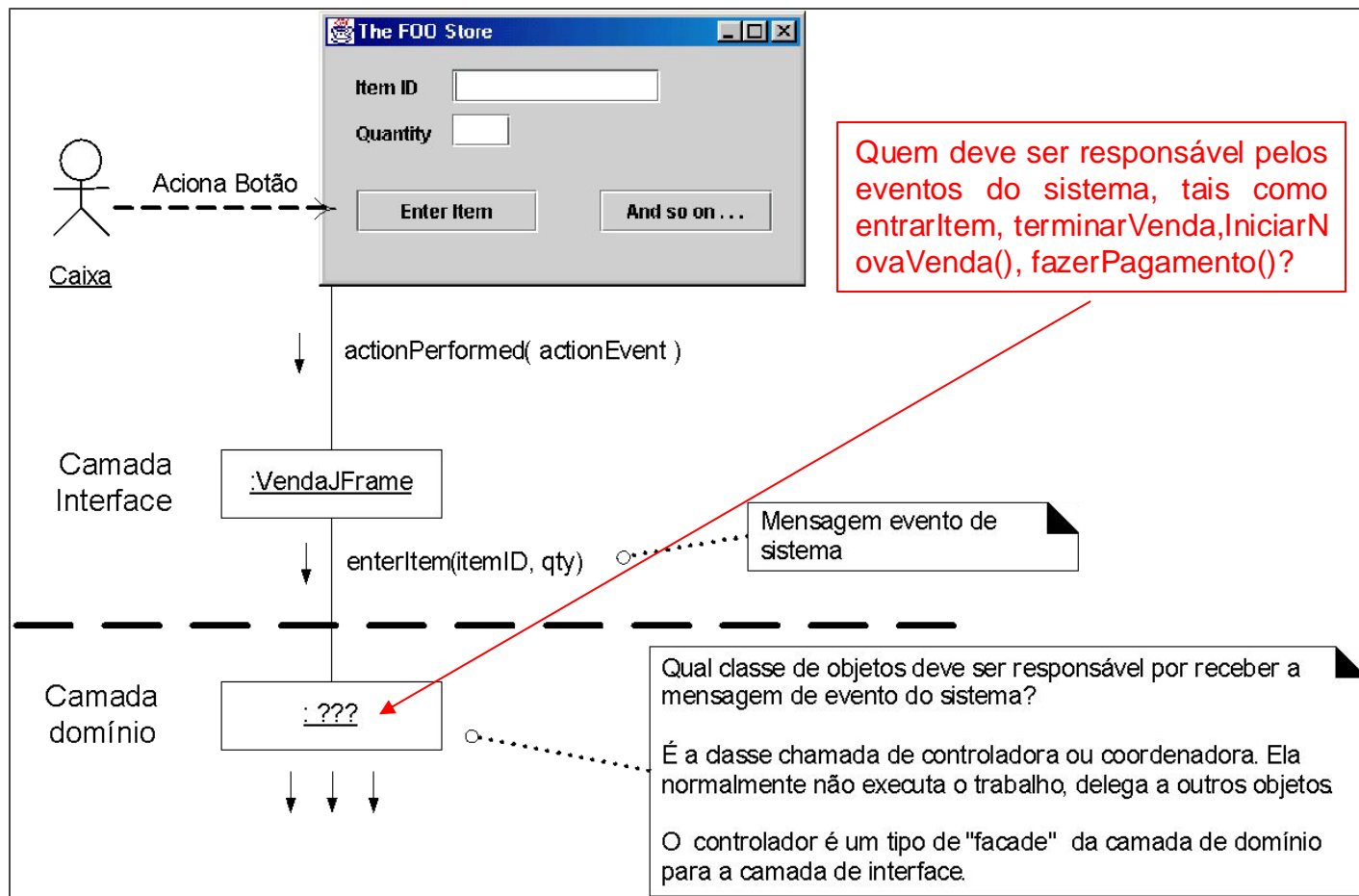
Projeto de Classes

Padrão Controlador

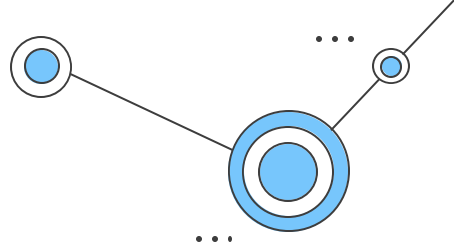
- Uma arquitetura em camadas simples tem uma camada de IU e uma camada de domínio entre outras.
- Tratar eventos na camada de aplicação.
- Objetos da IU devem apenas receber eventos, não tratá-los!



Padrão Controlador



Projeto de Classes



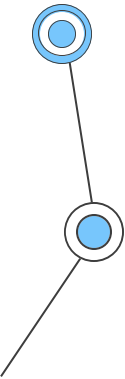
Padrão Controlador

Problema:

- Quem deve ser responsável por tratar um evento do sistema?

Solução:

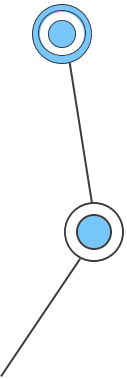
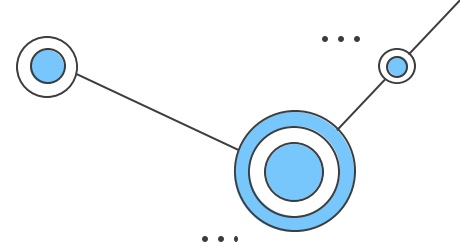
- Atribuir a responsabilidade de tratar um evento do sistema para uma classe “controladora” representando:
 - ✓ o sistema como um todo (facade controller).
 - ✓ o negócio ou organização com um todo (facade controller).
 - ✓ uma coisa ou papel de uma pessoa do mundo real envolvida diretamente com a tarefa (role controller).
 - ✓ um “tratador” (handler) artificial para todos os eventos de um caso de uso (use-case controller).



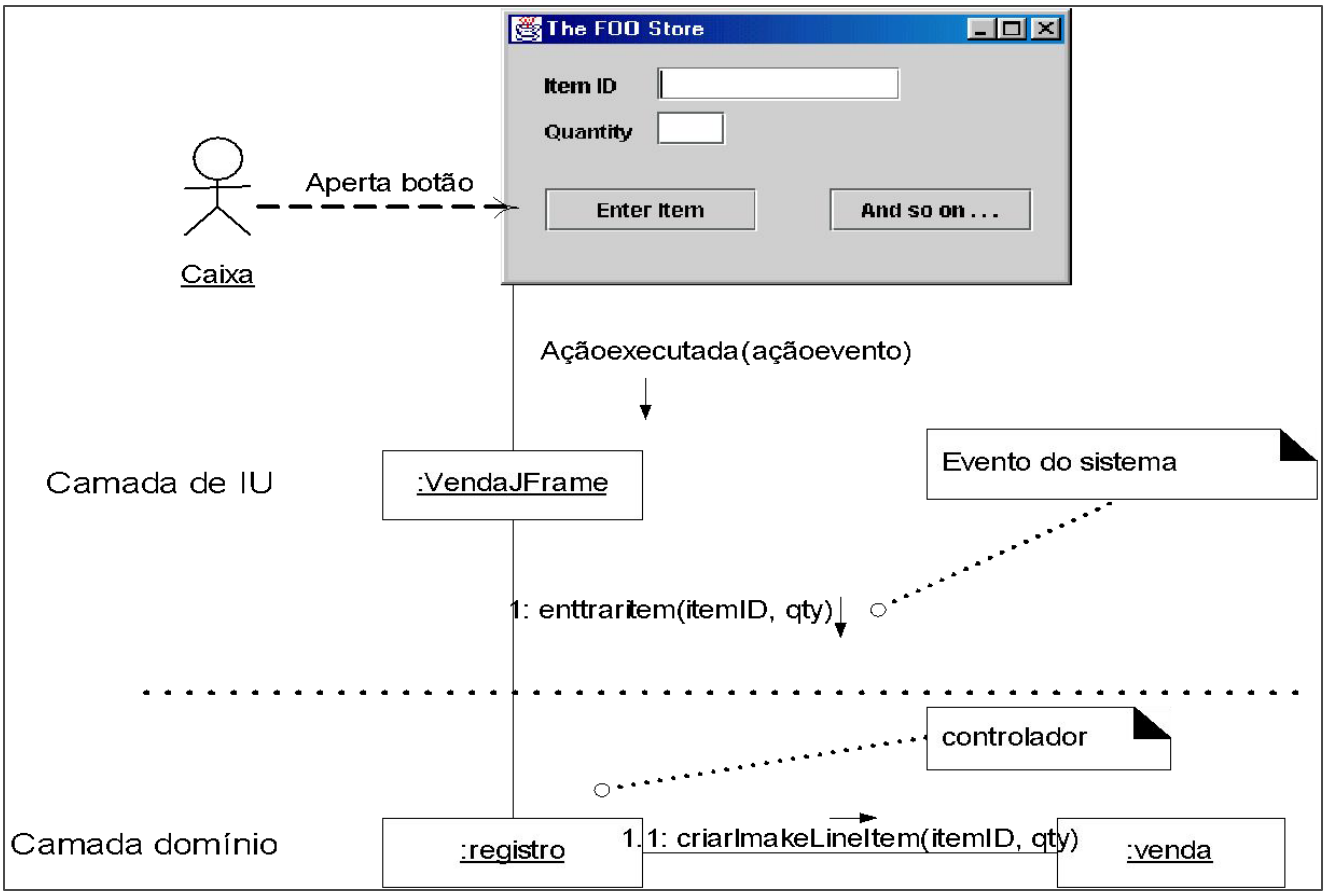
Projeto de Classes

Corolário Padrão Controlador

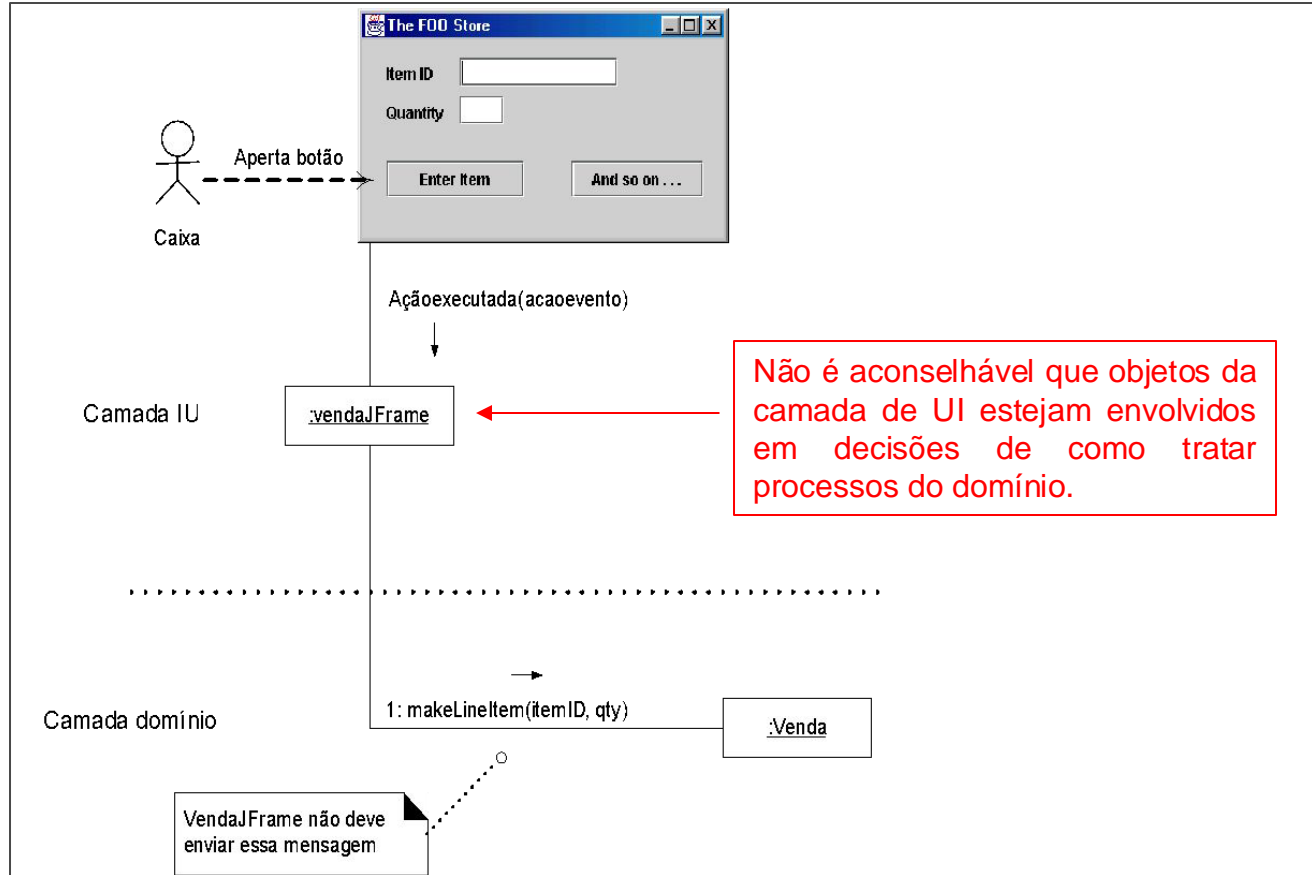
- objetos com interfaces externas (interface com o Usuário) e a camada de apresentação não devem ser responsáveis pelo atendimento a eventos do sistema.
- as operações do sistema devem ser tratadas na lógica da aplicação ou nas camadas de objetos do domínio.
- controladores são em geral objetos do lado cliente junto com a interface com o usuário.



Exemplo Corolário Padrão Controlador



Contra-Exemplo Corolário Padrão Controlador

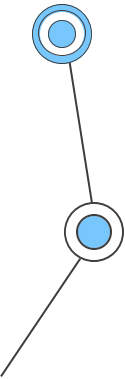
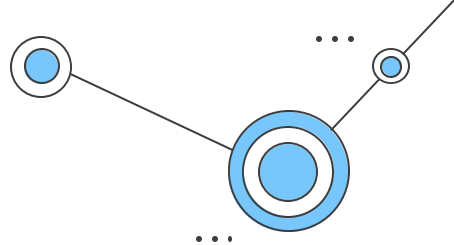


Projeto de Classes

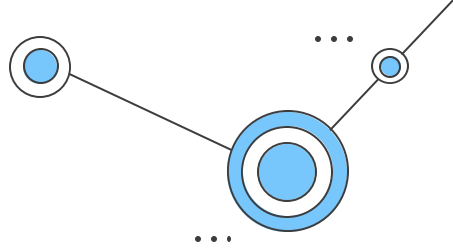
Padrão Controlador

Benefícios:

- aumento das possibilidades de reutilização e de interfaces plugáveis.
 - ✓ garante que a lógica da aplicação não seja tratada na camada de interface.
- conhecer o estado do caso de uso.
 - ✓ garantir que as operações do sistema ocorram em uma sequência válida.
 - Exemplo: `terminarVenda` deve preceder `fazerPagamento`.



Projeto de Classes



Padrão Controlador

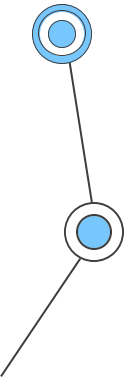
Considerações:

Se

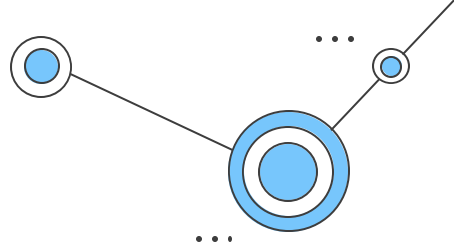
- ✓ Uma única classe recebe muitos ou todos eventos de sistema.
- ✓ o controlador executa muitas das tarefas necessárias para atender ao evento de sistema, sem delegar o trabalho.
- ✓ o controlador tem muitos atributos, e mantém informações sobre o sistema ou domínio que devem ser distribuídas para outros objetos.

Soluções:

- ✓ acrescentar mais controladores.
- ✓ projetar o controlador de forma que ele delegue o atendimento das responsabilidades de cada operação de sistema a outros objetos.



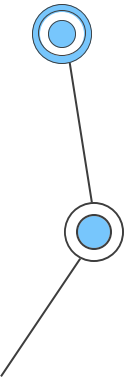
Projeto de Classes

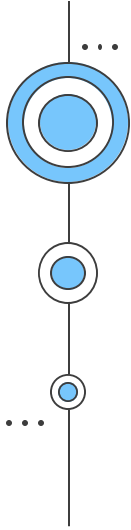


Padrão Controlador

Qual solução é melhor?

- Controlador Fachada
 - ✓ são adequados quando não existem muitos eventos de sistema muitos eventos podem levar a um controlador inchado, de baixa coesão e alto acoplamento.
- Controlador de casos de uso
 - ✓ são adequados quando o sistema possui muitos eventos com diferentes processos.
 - ✓ é necessário conhecer o estado de um caso de uso para identificar eventos fora de sequência Exemplo – corolário.

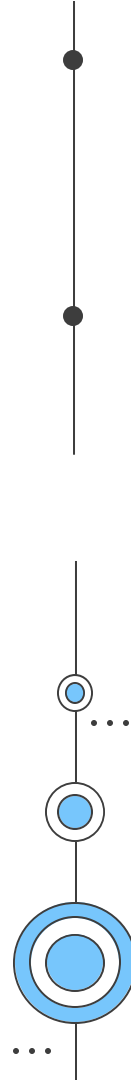


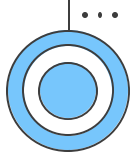


Projeto de Software

Referências básicas:

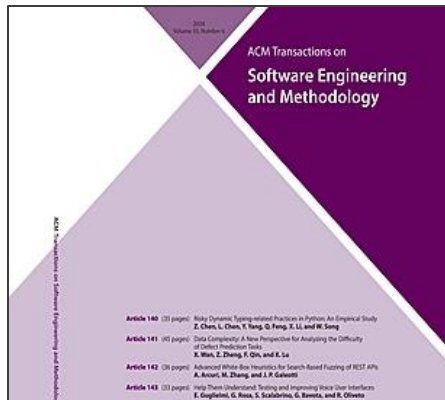
- **ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY**. New York, N.Y., USA: Association for Computing Machinery, 1992-. Trimestral. ISSN 1049-331X. Disponível em: <https://dl.acm.org/toc/tosem/1992/1/2>. Acesso em: 19 jul. 2024. (Periódico On-line).
- LARMAN, Craig. **Utilizando UML e padrões**: uma introdução á análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007. E-book. ISBN 9788577800476. (Livro Eletrônico).
- SILVEIRA, Paulo et al. **Introdução à arquitetura e design de software**: uma visão sobre a plataforma Java. Rio de Janeiro, RJ: Elsevier, Campus, 2012. xvi, 257 p. ISBN 9788535250299. (Disponível no Acervo).
- VERNON, Vaughn. **Implementando o Domain-Driven Design**. Rio de Janeiro, RJ: Alta Books, 2016. 628 p. ISBN 9788576089520. (Disponível no Acervo).



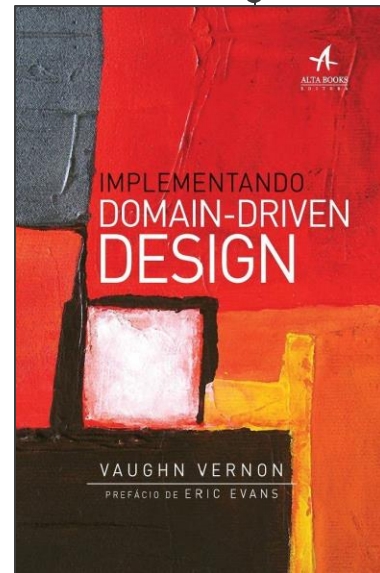
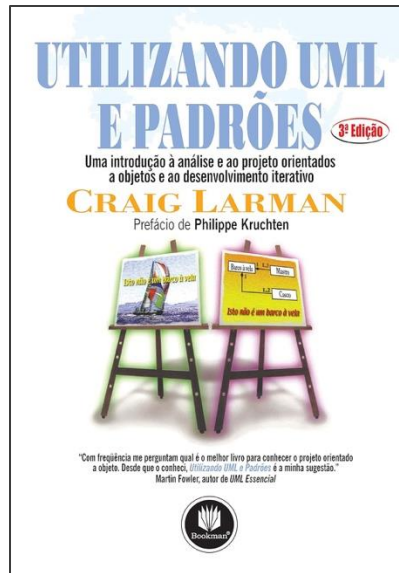


Projeto de Software

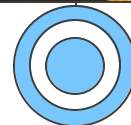
Referências básicas:

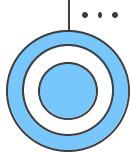


ACM Transactions on
Software Engineering
and Methodology



...



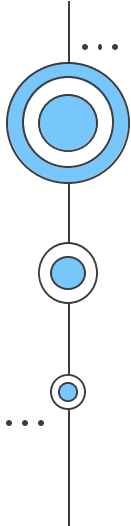


Projeto de Software

Referências complementares:

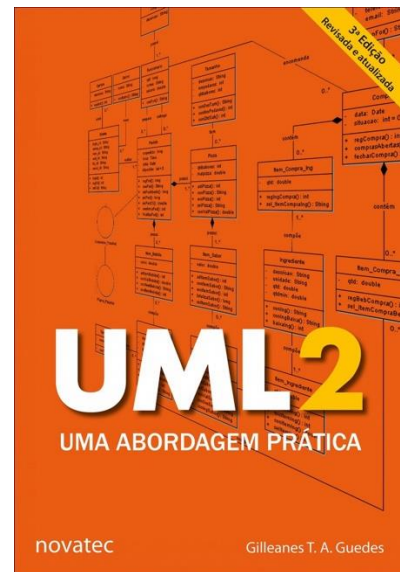
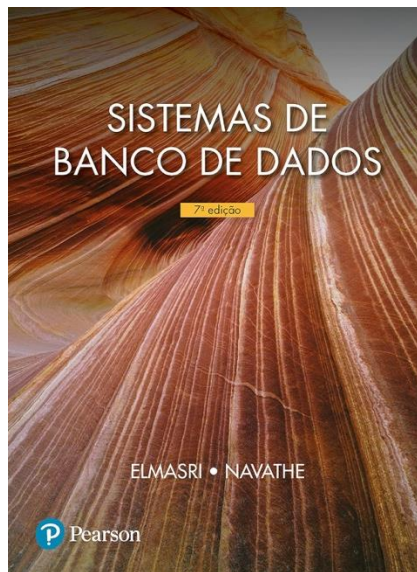
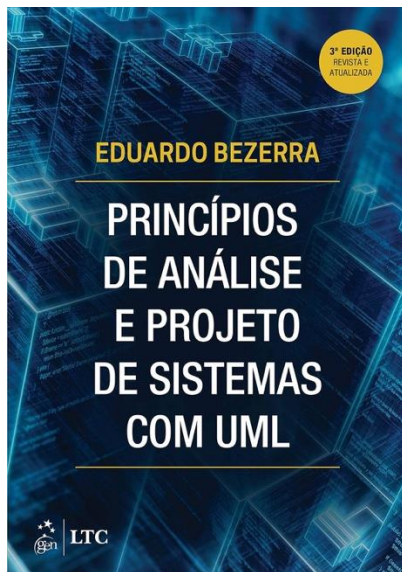
- BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 3. ed. rev. e atual. Rio de Janeiro: Elsevier, 2015. xvii, 398 p. ISBN 9788535226263. (Disponível no Acervo).
- ELMASRI, Ramez; Navathe, Shamkant B. **Sistemas de banco de dados**, 7ª ed. Editora Pearson 1152 ISBN 9788543025001. (Livro Eletrônico).
- GUEDES, Gilleanes T. A. **UML 2: uma abordagem prática**. 2. ed. São Paulo: Novatec, c2011. 484 p. ISBN 9788575222812. (Disponível no Acervo).
- **IEEE TRANSACTIONS ON SOFTWARE ENGINEERING**. New York: IEEE Computer Society, 1975-. Mensal,. ISSN 0098-5589. Disponível em: <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=32>. Acesso em: 19 jul. 2024. (Periódico On-line).
- SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, c2019. xii, 756 p. ISBN 9788543024974. (Disponível no Acervo).
- WAZLAWICK, Raul Sidnei. **Análise e design orientados a objetos para sistemas de informação: modelagem com UML, OCL e IFML**. 3. ed. Rio de Janeiro, RJ: Elsevier, Campus, c2015. 462 p. ISBN 9788535279849. (Disponível no Acervo).



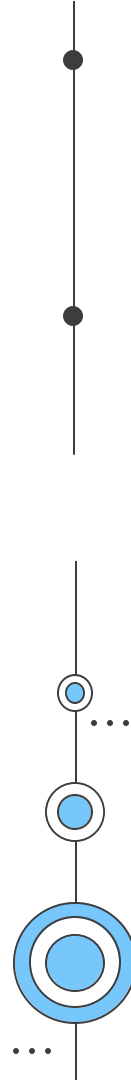


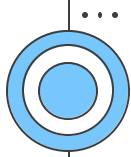
Projeto de Software

Referências complementares:



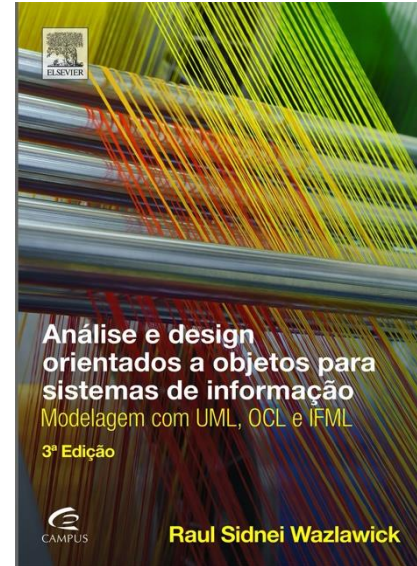
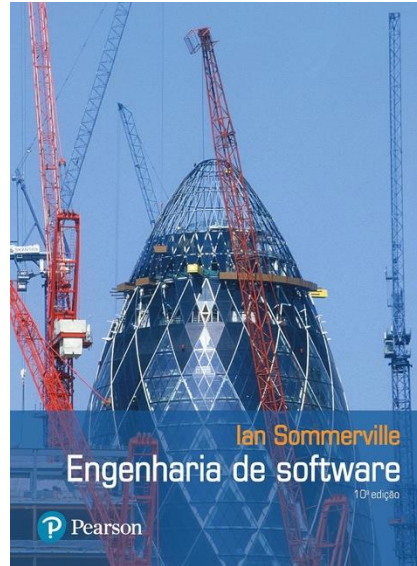
...





Projeto de Software

Referências complementares:



Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...