Curso - Padrões de Projeto
Módulo 1: Introdução

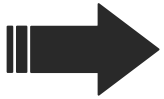
Vítor E. Silva Souza
vitorsouza@gmail.com

http://www.javablogs.com.br/page/engenho
http://esjug.dev.java.net

Sobre o Instrutor

- Formação:
  - Graduação em Ciência da Computação, com ênfase em Engenharia de Software, pela Universidade Federal do Espírito Santo (UFES);
  - Mestrado em Informática (em andamento) na mesma instituição.
- Java:
  - Desenvolvedor Java desde 1999;
  - Especialista em desenvolvimento Web;
  - Autor do blog Engenho – www.javablogs.com.br/page/engenho.
- Profissional:
  - Consultor em Desenvolvimento de Software Orientado a Objetos – Engenho de Software Consultoria e Desenvolvimento Ltda.

Estrutura do Curso



Módulo 1	Introdução
Módulo 2	Padrões de Criação
Módulo 3	Padrões de Estrutura
Módulo 4	Padrões de Comportamento
Módulo 5	O Padrão Model-View-Controller

Conteúdo deste módulo

- O que são e por que conhecê-los;
- Catálogos de padrões;
- Padrões de projeto na prática:
  - Como resolvem os problemas;
  - Como escolher e utilizar;
  - Frameworks.
- Anti-padrões.

Curso - Padrões de Projeto
Módulo 1: Introdução

O que são Padrões de Projeto e por que utilizá-los?

Desenvolvimento de Software

- Desenvolver sistemas OO reutilizáveis é muito difícil;
- Profissionais experientes fazem bons projetos, novatos têm dificuldade;
- Experiência consiste em utilizar técnicas que deram certo no passado.

Nascem os padrões

- Padrões são maneiras testadas e documentadas de alcançar objetivos;
- Padrões são comuns nas diversas áreas da engenharia;
- Na Engenharia de Software temos os Padrões de Projeto – Design Patterns.

O que são padrões de projeto

"Cada padrão descreve um problema que ocorre repetidas vezes no nosso ambiente e, então, descreve o núcleo da solução para aquele problema, de forma que você pode reutilizar a mesma solução milhões de vezes sem nunca fazê-la da mesma forma duas vezes."

Christopher Alexander, et. al

A Pattern Language (1977)

O que são padrões de projeto

"Descrição de uma solução customizada para resolver um problema genérico de projeto em um contexto específico. [...] Um padrão de projeto dá nome, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la reutilizável."

Erich Gamma, et. al

Design Patterns, Elements of Reusable Object-Oriented Software (1994)

Como é formado um padrão

- Quatro elementos essenciais:
  - Nome;
  - Descrição do problema e contexto para os quais o padrão se aplica;
  - Descrição da solução genérica proposta;
  - Consequências da aplicação do padrão (custos e benefícios).

Por que usar padrões

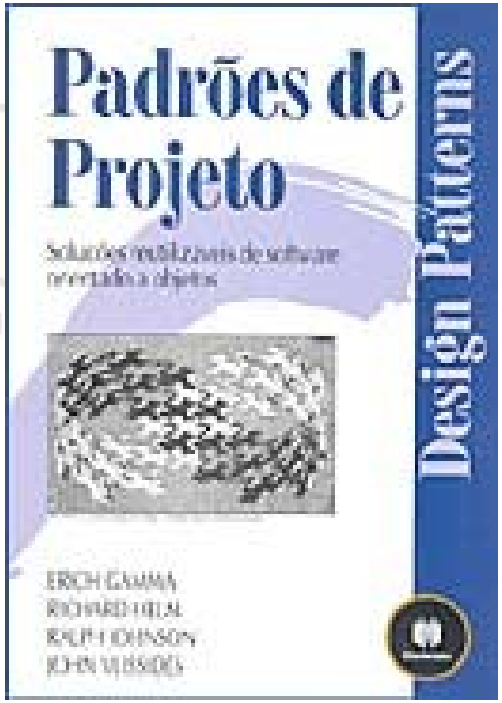
- Aprender com a experiência de outros:
  - Ajudam a resolver os principais problemas dos iniciantes;
  - Permitem que façamos bons projetos mais rapidamente.
- O jargão facilita a comunicação;
- Software de melhor qualidade:
  - Melhores práticas em OO;
  - Soluções bem testadas e documentadas.

Curso - Padrões de Projeto
Módulo 1: Introdução

Catálogos de Padrões de Projeto: classificação e resumo dos padrões mais conhecidos.

Catálogos de padrões

- Registram as experiências bem-sucedidas de um grupo de pessoas:
  - Livro "Design Patterns"
    Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four);
  - Padrões J2EE da Sun
    http://java.sun.com/blueprints/patterns/
  - Dentre outros...



Descrição dos padrões do GoF

- Nome e classificação;
- Intenção;
- Outros nomes;
- Motivação;
- Aplicabilidade;
- Estrutura;
- Participantes;
- Colaborações;
- Consequências;
- Implementação;
- Código de exemplo;
- Usos conhecidos;
- Padrões relacionados.

Organização do catálogo do GoF

Escopo \ Propósito	Criação	Estrutura	Comportamento
Classe	Factory Method	Adapter (classe)	Interpreter Template Method
Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Organização do catálogo do GoF

- Propósito:
  - Criacional: processo de criação de objetos;
  - Estrutural: composição de classes e objetos;
  - Comportamental: interação e distribuição de responsabilidades entre objetos e classes;
- Escopo:
  - Classe: relação entre classes e subclasses (herança);
  - Objeto: relação entre objetos (associação, composição).

Padrões de Criação

- Factory Method (Método Fábrica):
  - Define uma interface para criação de objetos mas deixa a implementação para as subclasses.
- Abstract Factory (Fábrica Abstrata):
  - Provê uma interface para criação de famílias de objetos relacionados sem especificar suas classes concretas.
- Builder (Construtor):
  - Separa o processo de construção de objetos complexos, desacoplando a criação de instâncias destes objetos.

Padrões de Criação

- Prototype (Protótipo):
  - Define um protótipo dos objetos a serem usados e cria-os clonando este protótipo.
- Singleton (Objeto Único):
  - Garante que uma classe possui somente uma instância e provê um ponto de acesso global a ela.

Padrões de Estrutura

- Adapter (Adaptador):
  - Converte uma interface de uma classe em outra que objetos clientes esperam utilizar.
- Bridge (Ponte):
  - Desacopla uma abstração de sua implementação para que ambas possam variar independentemente.
- Composite (Composto):
  - Agrupa objetos de mesma interface em estrutura de árvore para tratar todos eles como se fossem uma única entidade.

Padrões de Estrutura

- Decorator (Decorador):
  - Estende a funcionalidade de um objeto dinamicamente anexando objetos que irão executar antes ou depois do objeto "decorado".
- Façade (Fachada):
  - Provê uma interface única (ponto central) para diversas outras interfaces do sistema.

Padrões de Estrutura

- Flyweight (Peso Mosca):
  - Forma um pool de objetos imutáveis para serem utilizados em diversas partes do sistema.
- Proxy (Procurador):
  - Provê um intermediário (procurador) que representa o objeto mas se comporta de forma diferente.

Padrões de Comportamento

- Interpreter (Interpretador):
  - Criar uma linguagem de representação de operações e construir um interpretador para essa linguagem.
- Template Method (Método Modelo):
  - Define o esqueleto do algorítmo da operação na superclasse, delegando partes do mesmo para as subclasses.
- Chain of Responsibility (Cadeia de Repons.):
  - Monta uma corrente de objetos que serve a uma requisição, dando a cada um a oportunidade de respondê-la e/ou passá-la adiante.

Padrões de Comportamento

- Command (Comando):
  - Encapsula requisições como objetos, permitindo parametrização, log ou *undo* de funções.
- Iterator (Iterador):
  - Acessar diversos elementos de um conjunto em sequência sem expor sua representação interna.
- Mediator (Mediador):
  - Delega a um objeto a responsabilidade de fazer outros objetos se comunicarem, tirando destes últimos o acoplamento entre si.

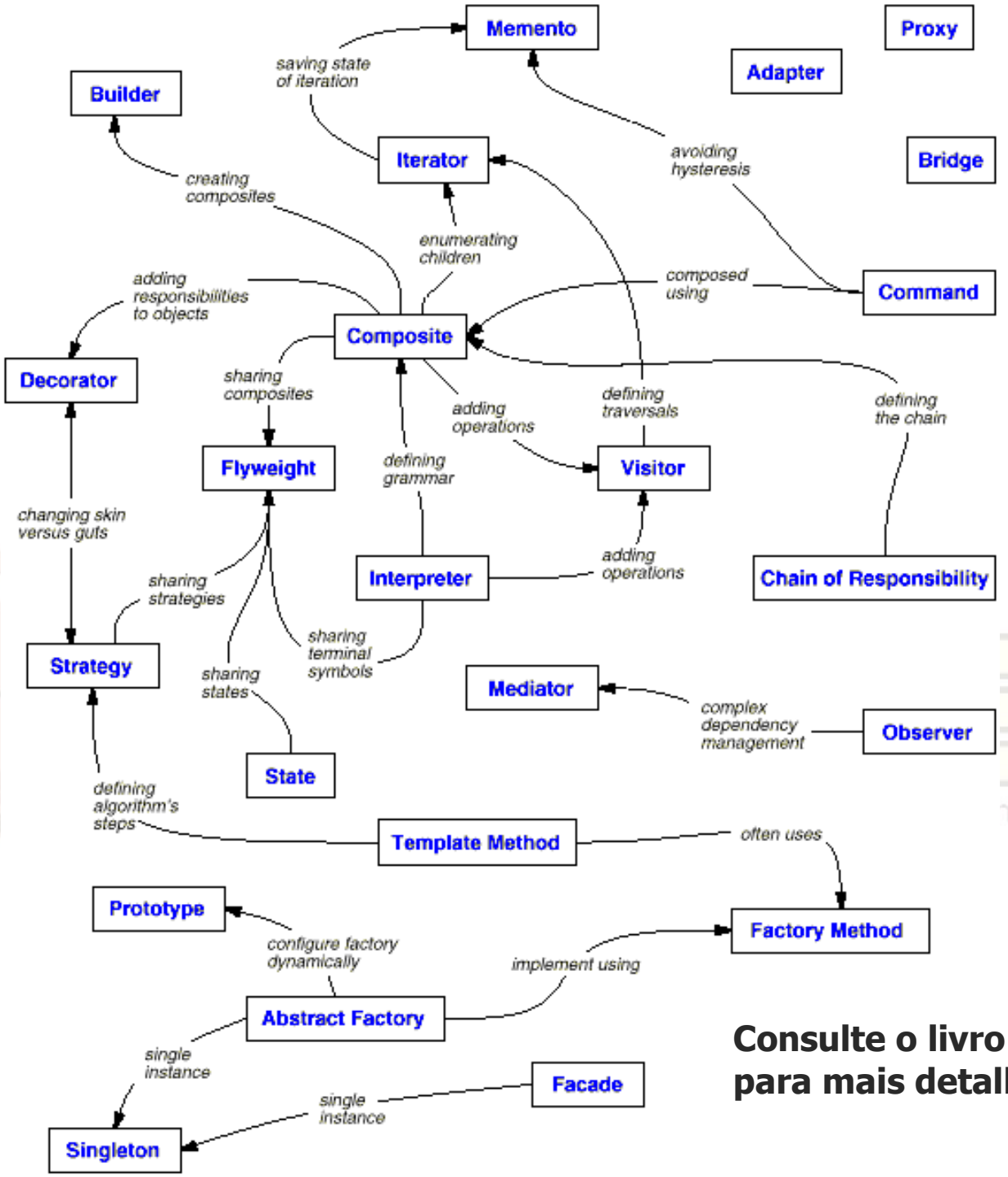
Padrões de Comportamento

- Memento (Recordação):
  - Sem violar o encapsulamento, armazena o estado atual do objeto para que possa ser restaurado posteriormente.
- Observer (Observador):
  - Define uma dependência um-para-muitos entre objetos tal que quando um objeto muda de estado, todos são notificados.
- State (Estado):
  - Permite que um objeto altere seu comportamente quando mudar seu estado.

Padrões de Comportamento

- Strategy (Estratégia):
  - Define uma família de algorítmos, encapsula-os em objetos e permite que sejam intercambiados.
- Visitor (Visitante):
  - Representa operações em objetos como outros objetos com uma interface comum, permitindo que sejam criadas novas operações sem alterar o objeto.

Relacionamento entre os Padrões



Consulte o livro do GoF para mais detalhes.

Outros padrões

- GoF é a referência clássica;
- Novos problemas incitam a criação de novos padrões de projeto;
  - Muitos utilizam ou são baseados nos padrões já descritos no catálogo do GoF.
- É importante conhecê-los porque se referem a domínios mais específicos:
  - Ex.: um desenvolvedor J2EE deve conhecer os padrões de projeto J2EE.

Outros padrões – exemplos:

- Injeção de Dependências (ou IoC):
  - Tira a responsabilidade do objeto encontrar sua dependência, que é automaticamente injetada por um objeto externo.
- Model – View – Controller:
  - Utiliza Observer, Composite e Strategy para definir uma divisão entre dados (modelo), interface com o usuário (visão) e a lógica de controle.

Outros padrões – exemplos:

- Value Object (VO / DTO):
  - Cria uma cópia do objeto de domínio com algumas informações necessárias e envia-a como parâmetro no lugar do objeto original.
- Data Access Object (DAO):
  - Delega a uma classe específica toda a lógica de acesso a dados persistentes (banco de dados).

Curso - Padrões de Projeto
Módulo 1: Introdução

Padrões de Projeto na Prática: como os padrões resolvem problemas, como selecioná-los e utilizá-los.

Determinando os objetos mais apropriados

- Decompor um sistema em objetos é difícil. Várias questões:
  - Encapsulamento, granularidade, dependências, flexibilidade, desempenho, evolução, reusabilidade, etc.
- Padrões de projeto podem ajudar:
  - Identificação de abstrações menos óbvias de conceitos não presentes na natureza, como um algorítmo ou um estado.

Determinando a granularidade ideal

- Um objeto pode representar uma aplicação inteira ou um detalhe minúsculo. Como escolher?
- Padrões de projeto podem ajudar:
  - Façade explica como representar sistemas inteiros num só objeto;
  - Flyweight descreve como utilizar muitos objetos de pequena granularidade;
  - Abstract Factory, Builder, Visitor e Command descrevem formas específicas de composição de objetos.

Especificando interfaces

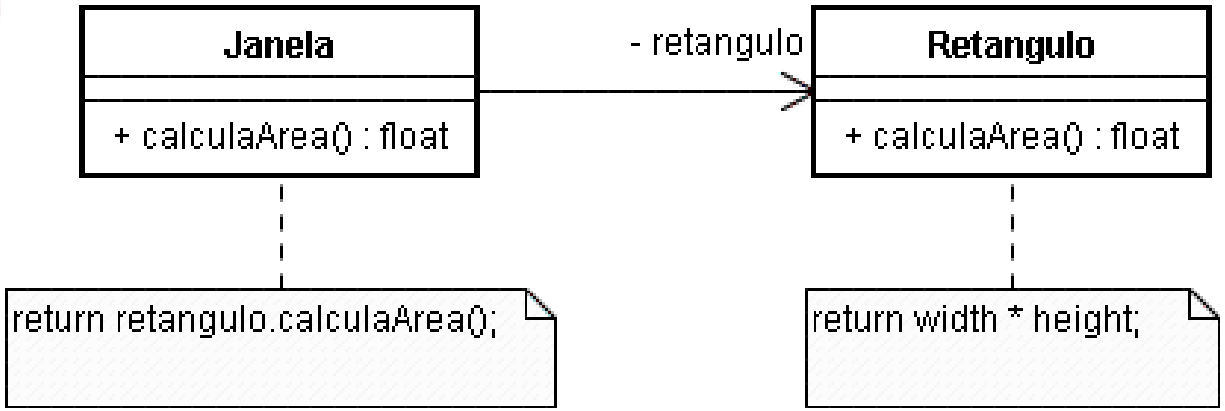
- Interfaces:
  - Conjunto de mensagens que um objeto aceita (contrato, assinaturas);
  - Objetos são conhecidos pelas interfaces, que nada dizem sobre a implementação.
- Padrões de projeto podem ajudar:
  - Identificam os elementos-chave de interfaces;
  - Restringem o que incluir em interfaces;
  - Estabelecem relacionamentos entre elas.

Especificando implementações

- Boa prática: “programe para interfaces, não para implementações”;
- Padrões de projeto podem ajudar:
  - Abstraem o processo de criação de objetos, desacoplando os objetos clientes de implementações, permanecendo somente as interfaces.

Tirando proveito do reuso

- Como construir software reutilizável?
  - "Dê preferência à composição de objetos em detrimento à herança de classes".
- Padrões de projeto podem ajudar:
  - Delegação dá mais poder à composição.



Relacionando estruturas em tempo de compilação e execução

- A estrutura runtime é bem diferente do código: dinamismo e interação;
  - Ex.: agregação e associação são codificadas da mesma forma, mas são bem diferentes em *runtime*.
- Padrões de projeto podem ajudar:
  - Se você entende o padrão, passa a entender a distinção, que está explícita na documentação do padrão.

Projetando para mudanças

- Sistemas devem ser projetados para facilitarem a manutenção:
  - Riscos de custos imprevistos;
  - Retrabalho.
- Padrões de projeto podem ajudar:
  - Promovem desacoplamento, permitindo maior liberdade dos objetos;
  - Código mais robusto, legível e de maior qualidade.

Causas de retrabalho e padrões (GoF) que as evitam

1. Criação de objeto especifica classe explicitamente:
   - O sistema está preso a uma implementação específica;
   - Solução: criar objetos indiretamente com Abstract Factory, Factory Method ou Prototype.
2. Dependência em operações específicas:
   - O sistema só tem uma forma de satisfazer a uma requisição;
   - Solução: evitar requisições "hard-coded" com Chain of Responsibility ou Command.

Causas de retrabalho e padrões (GoF) que as evitam

1. Dependência de plataforma:
   - O software utiliza recursos específicos de uma plataforma;
   - Solução: limitar dependências com Abstract Factory ou Bridge.
2. Dependência em representações ou implementações de objetos:
   - Clientes que sabem como um objeto é implementado, representado ou armazenado podem ter que ser alterados se o objeto mudar;
   - Solução: isolar os clientes com Abstract Factory, Bridge, Memento ou Proxy.

Causas de retrabalho e padrões (GoF) que as evitam

1. Dependência de algorítmo:
   - Objetos que dependem de algorítmos precisam mudar quando o algorítmo mudar;
   - Solução: isolar os algorítmos com Builder, Iterator, Strategy, Template Method ou Visitor.
2. Forte acoplamento:
   - Classes fortemente acopladas são difíceis de reusar, testar, manter, etc. Sistema monolítico;
   - Solução: enfraquecer o acoplamento com Abstract Factory, Bridge, Command, Chain of Responsibility, Façade, Mediator ou Observer.

Causas de retrabalho e padrões (GoF) que as evitam

1. Extensão de funcionalidade por meio de subclasse:
   - Herança é difícil de usar, composição dificulta a compreensão;
   - Solução: usar padrões que implementem bem herança, composição e delegação como Bridge, Chain of Responsibility, Composite, Decorator, Observer ou Strategy.
2. Incapacidade de alterar classes convenientemente:
   - Classes inacessíveis, imcompreensíveis ou difíceis de alterar;
   - Solução: usar Adapter, Decorator ou Visitor.

Como selecionar um padrão

- Entenda como os padrões ajudam a resolver problemas em OO;
- Revise as intenções de cada padrão;
- Estude como os padrões se inter-relacionam;
- Estude as similaridades entre os padrões de mesmo propósito;
- Conheça as principais causas de retrabalho;
- Considere o que você pode querer mudar em seu projeto no futuro.

Resumo do Catálogo do GoF

Padrões de Criação

Padrão	Aspectos que podem variar
Abstract Factory	Famílias de objetos "produto".
Builder	Como um objeto composto é criado.
Factory Method	Subclasse do objeto que é instanciado.
Prototype	Classe do objeto que é instanciado
Singleton	Uma única instância da classe.

Resumo do Catálogo do GoF

Padrões de Estrutura

Padrão	Aspectos que podem variar
Adapter	Inteface para um objeto.
Bridge	Implementação de um objeto.
Composite	Estrutura e composição de um objeto.
Decorator	Responsabilidades de um objeto sem recorrer à herança.
Façade	Interface para um subsistema.
Flyweight	Custo de armazenamento de objetos.
Proxy	Como um objeto é acessado, sua localização.

Resumo do Catálogo do GoF

Padrões de Comportamento

Padrão	Aspectos que podem variar
Chain of Responsibility	Objeto que pode responder a uma requisição.
Command	Onde e como uma requisição é respondida.
Interpreter	Gramática e interpretação de uma linguagem.
Iterator	Como elementos agregados são acessados.
Mediator	Como e quais objetos interagem uns com os outros.

Resumo do Catálogo do GoF

Padrões de Comportamento

Padrão	Aspectos que podem variar
Memento	Quais informações privadas são armazenadas fora do objeto e quando.
Observer	Número de objetos que dependem de outro e como estes ficam atualizados.
State	Estados de um objeto.
Strategy	Um algoritmo.
Template Method	Passos de um algoritmo.
Visitor	Operações que podem ser aplicadas a objetos sem alterar suas classes.

Como usar um padrão

1. Leia o padrão todo uma vez;
2. Estude o código fonte de exemplo;
3. Escolha nomes para os participantes do padrão dentro do seu contexto;
4. Defina as novas classes e modifique classes existentes que são afetadas;
5. Defina nomes para as operações do padrão dentro do seu contexto;
6. Implemente as operações.

Curso - Padrões de Projeto
Módulo 1: Introdução

Anti-Padrões: soluções que
você deve evitar.

O que é um anti-padrão

- Antipatterns são soluções que trazem mais complicação do que benefício;
- Pode advir de:
  - Uso incorreto de padrões de projeto;
  - Criação de um novo padrão sem as devidas reflexões e testes;
  - O que era bom ontem pode não ser mais.
- É necessário entendê-los para não cair em armadilhas e, se cair, saber se recuperar delas.

Exemplos de anti-padrões

- Analysis paralysis;
- Stovepipe system;
- Design by committee;
- Copy and paste programming;
- Spaghetti code;
- God object/class;
- Vendor lock-in;
- E muitos outros...

Para saber mais...

- Neste curso iremos focar nos padrões de projeto, e não nos anti-padrões;
- Para saber mais:
  - Wikipedia
    http://en.wikipedia.org/wiki/Antipattern
  - Antipatterns.com:
    http://www.antipatterns.com/

Curso - Padrões de Projeto
Módulo 1: Introdução

Vítor E. Silva Souza
vitorsouza@gmail.com

http://www.javablogs.com.br/page/engenho
http://esjug.dev.java.net