AULA 11: Monitoramento da Qualidade Interna: Padrões.

Professora: **Rosana T. Vaccare Braga**

Agenda

- Conceito de monitoramento da qualidade interna
- Padrões
- Ferramentas e tecnologias

Relembrando: Conceito de monitoramento da qualidade interna

- **Avaliação sistemática dos atributos e métricas internas do código, para identificar possíveis falhas, inconsistências ou oportunidades de melhoria na estrutura do software.**
- **É fundamental para garantir a qualidade e a confiabilidade do produto final, bem como para aprimorar o processo de desenvolvimento de software de forma contínua.**
  - **é possível identificar problemas precocemente, definir objetivos claros de qualidade, planejar melhorias no código-fonte e acompanhar os resultados alcançados.**

Conceito de monitoramento da qualidade interna

- **Algumas formas de monitorar a qualidade interna:**
  - **métricas de software**
  - **convenção de nomenclatura**
  - **estratégias de clean code**
    - **Princípios de código limpo**
    - **Técnicas de refatoração**
    - **Análise de código estático**
    - **Práticas de desenvolvimento de equipe**
    - **adoção de padrões de implementação, de projeto, etc.**
    - **Ferramentas e tecnologias**
  - **e outros**

Aula de hoje

Introdução à qualidade interna de software

- **Quais são as métricas comuns usadas para avaliar a qualidade interna de software?**
  - **Conformidade com padrões:** mede o grau em que um sistema adere a padrões de codificação ou práticas recomendadas.
    - Para Sommerville, alguns padrões, como os de interface de usuário, podem ser implementados como um conjunto de componentes reusáveis.
      - Por exemplo, se os menus em uma interface de usuário forem implementados usando componentes reusáveis, todas as aplicações apresentarão os mesmos formatos de menu para os usuários. O uso de interfaces de usuário-padrão melhora a confiança, pois os usuários cometem menos erros quando são apresentados a interfaces familiares.

Padrões de software

- **São soluções comprovadas para problemas comuns encontrados no processo de desenvolvimento de software (Gamma et al, 1994).**
- **Eles são soluções para problemas de análise, arquitetura, projeto, codificação, manutenção (entre outros) que são documentados de forma que possam ser reutilizados por outros desenvolvedores para solucionar problemas semelhantes.**
- **No caso de padrões de implementação ajudam a melhorar a qualidade do código, reduzem a complexidade do software e aceleram o processo de desenvolvimento.**
  - **Em outras palavras, os padrões são uma forma de capturar e comunicar a experiência adquirida na criação de software, e de tornar essa experiência disponível para outros desenvolvedores (BECK, 2013).**

Gamma et al., Design Patterns 1994.

BECK, Kent. Padrões de Implementação. 2013.

Padrões de implementação

- **Principais padrões de implementação:**
  - **Padrões de projeto: conjunto de soluções comuns para problemas recorrentes no desenvolvimento de software, que ajudam a garantir a qualidade interna do código e a facilitar sua manutenção.**
  - **Boas práticas de codificação: conjunto de orientações para a escrita de código limpo e legível, que facilita a manutenção e a evolução do software.**

Vejam aula 10

Kent Beck. Padrões de Implementação.
Ian Sommerville. 10ª Edição. Engenharia de Software.

Padrões de Projeto

- **O que são e como eles podem ajudar a melhorar a qualidade interna do software?**
  - São soluções comuns para problemas de desenvolvimento de software que foram identificados e documentados ao longo do tempo. Eles são considerados boas práticas em engenharia de software e ajudam a garantir a qualidade interna do software, tornando-o mais fácil de entender, modificar e manter.
  - Exemplo: o padrão de projeto MVC (Model-View-Controller) pode ajudar a separar as responsabilidades de apresentação, processamento de dados e controle de fluxo em uma aplicação web.

Padrões de Projeto

- Os padrões de projeto tiveram um enorme impacto no projeto de software orientado a objetos.
  - Os padrões são soluções já testadas para problemas comuns e se tornaram um vocabulário para falar sobre o projeto.
  - É possível explicar um projeto por meio das descrições dos padrões usados, especialmente os padrões de projeto mais conhecidos descritos pelo livro da 'Gangue dos Quatro' (GAMMA et al., 1995).

Padrões de Projeto

- **Quais são os padrões de projeto mais comuns?**
  - **Singleton:** o padrão Singleton é usado para garantir que uma classe tenha apenas uma única instância em todo o sistema. Isso é útil quando você precisa limitar o acesso a um recurso compartilhado.
  - **Observer:** o padrão Observer é usado para permitir que um objeto observe as mudanças de estado de outro objeto. Isso é útil quando você precisa atualizar vários objetos quando um objeto muda de estado.
  - **Strategy:** o padrão Strategy é usado para encapsular um algoritmo dentro de uma classe. Isso permite que você modifique o algoritmo sem afetar o restante do código.

Padrões de Projeto

- **Objeto Unitário (Singleton)**
  - Utilizado quando é necessário garantir que uma classe possui apenas uma instância, que fica disponível às aplicações-cliente de alguma forma.
    - Por exemplo, uma base de dados é compartilhada por vários usuários, mas apenas um objeto deve existir para informar o estado da base de dados em um dado momento.

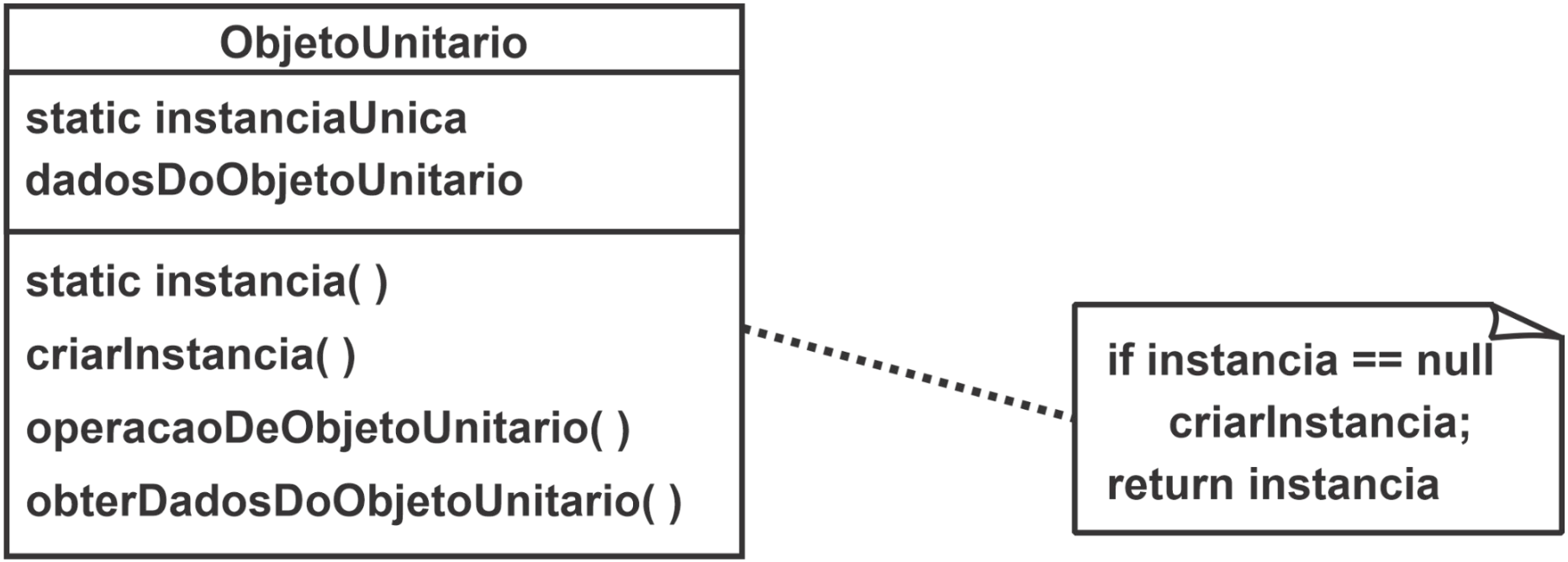
Padrões de Projeto

- **Objeto Unitário**
  - **Problema**
    - **Como garantir que uma classe possui apenas uma instância e que ela é facilmente acessível?**
  - **Forças**
    - **Uma variável global poderia ser uma forma de tornar um objeto acessível, embora isso não garanta que apenas uma instância seja criada.**

- **Objeto Unitário**
  - **Solução**
    - **Fazer a própria classe responsável por controlar a criação de uma única instância e de fornecer um meio para acessar essa instância.**
    - **A classe Objeto Unitário define um método instância para acesso à instância única, que verifica se já existe a instância, criando-a se for necessário.**
      - **Na verdade esse é um método da classe (static em C++ e Java), ao invés de método do objeto.**
      - **A única forma da aplicação-cliente acessar a instância única é por meio desse método.**
      - **Construtor é privado e instância das classes só poderão ser obtidas por meio da operação pública e estática *getInstance()*.**

Padrões de Projeto

- **Objeto Unitário**



Padrões de Projeto

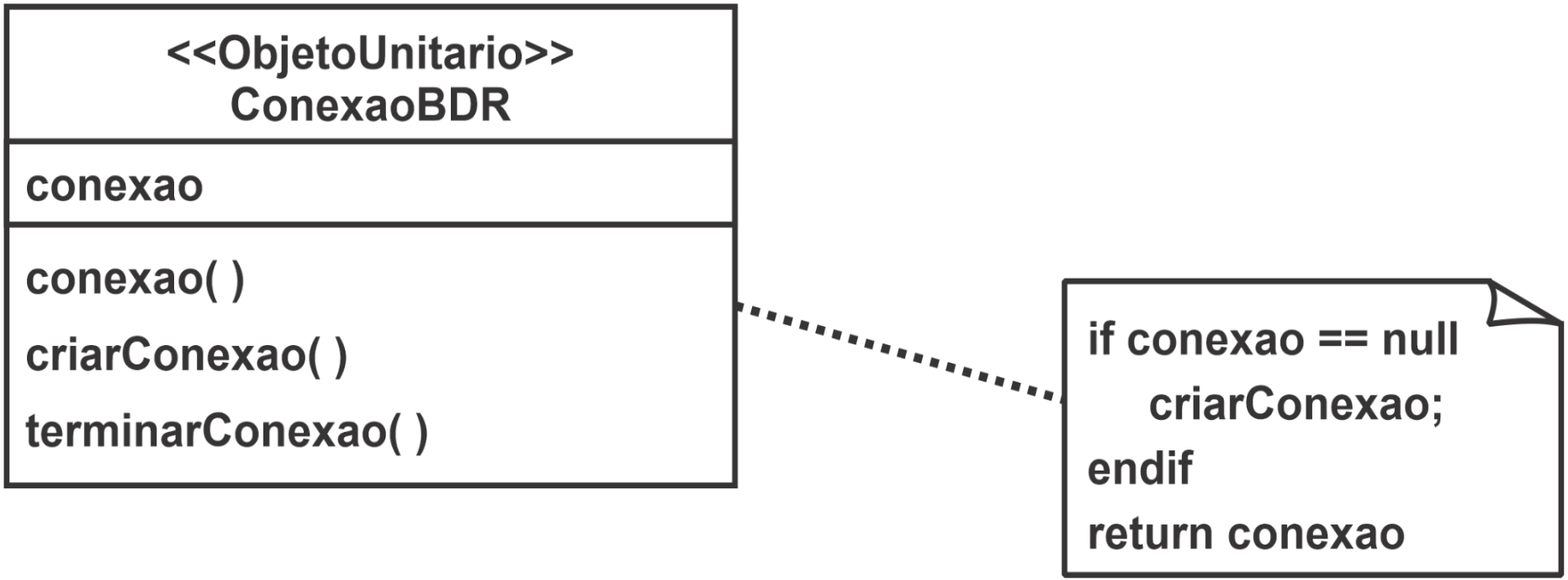
- **Exemplo de aplicação de padrões de projeto**
  - **Problema:** Ao iniciar uma aplicação, será necessário estabelecer uma conexão com a base de dados, para ter um canal de comunicação aberto durante a execução da aplicação.
    - Em geral, isso é implementado por meio de uma classe Conexão (várias interfaces de BDRs existentes fornecem uma classe Connection especialmente para este fim).
    - Esta conexão deve posteriormente ser fechada, no momento do término da aplicação.

Padrões de Projeto

- **Exemplo de aplicação de padrões de projeto**
  - **A conexão com o BDR deve preferencialmente ser única, pois se cada vez que um objeto for criado uma nova conexão for criada, inúmeros objetos existirão na memória, desnecessariamente.**
  - **Assim, pode-se aplicar o padrão Objeto Unitário para garantir que, para cada aplicação sendo executada, uma única conexão válida com o BDR existe.**

Padrões de Projeto

- **Exemplo de aplicação de padrões de projeto**

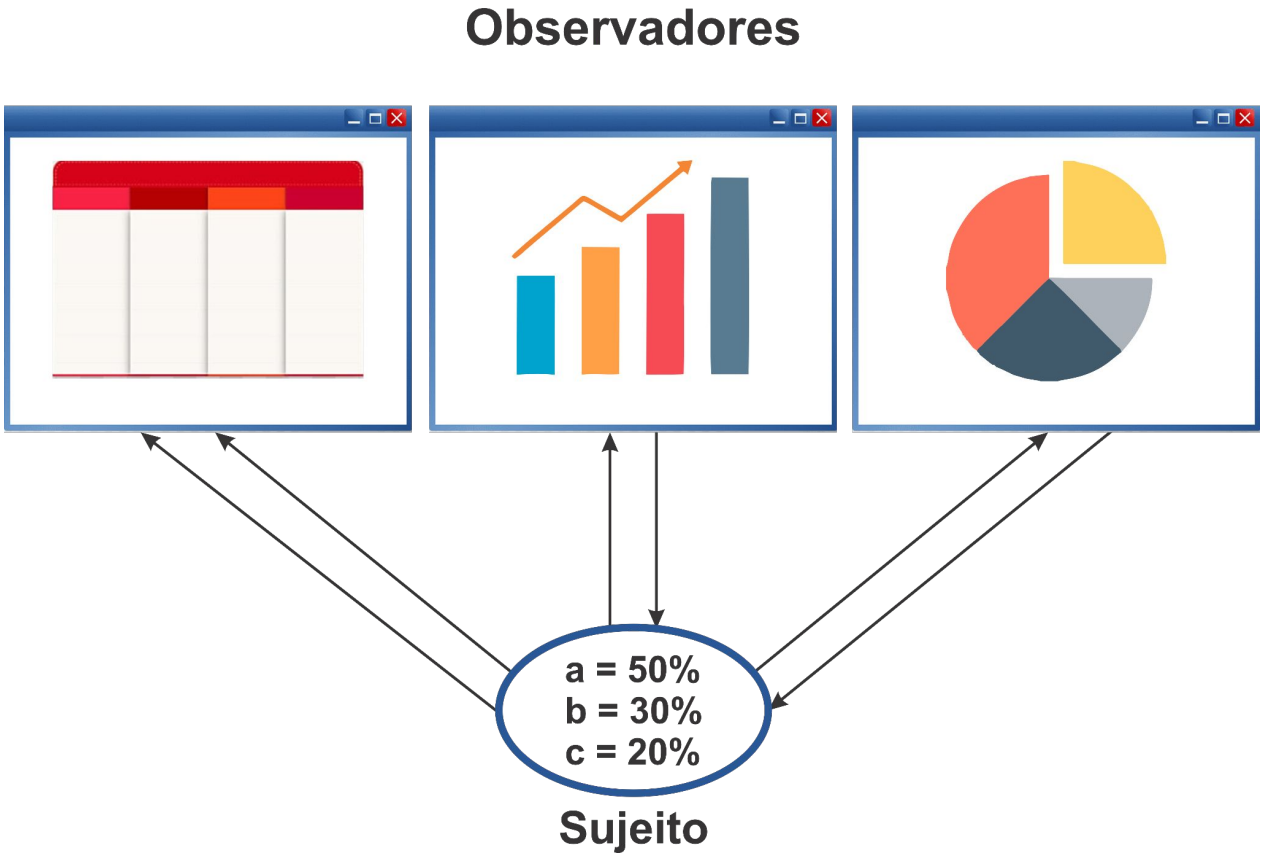


Padrões de Projeto

- **Observador (Observer)**
  - **Aplicação:** sistemas cujas informações são preferencialmente centralizadas em uma única fonte (sujeito), e existem diversas visões (observadores) que devem ser atualizadas sempre que a informação for modificada
  - É uma generalização do padrão MVC (Model-view-controller)

Padrões de Projeto

- **Observador (Observer)**



Padrões de Projeto

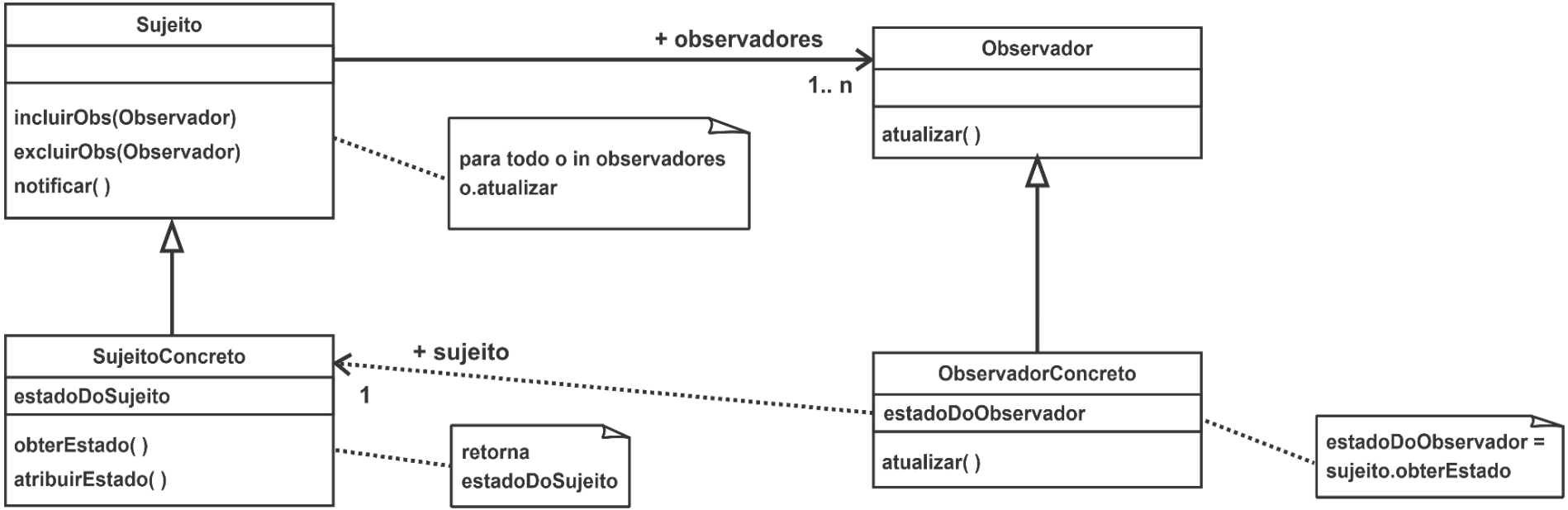
- **Observador (Observer)**
  - **Problema:** Quando se utiliza orientação a objetos, o sistema é dividido em um conjunto de classes que interagem, mas é difícil manter a consistência entre objetos relacionados.
  - **Forças:** Qualquer tentativa de manter a consistência aumenta o acoplamento entre as classes.
    - Por exemplo, em uma ferramenta integrada, pode-se separar os dados em si das diversas visões sobre esses dados, de forma que mudança em uma das visões seja refletida nas demais, mesmo que as visões não tenham conhecimento umas das outras.
      - Essa separação pode levar a um projeto mais fácil de entender, estender, manter e reusar cada objeto separado.

Padrões de Projeto

- **Observador (Observer)**
  - **Solução:**
    - **Separar em objetos diferentes as informações e suas visões.**
    - **Definir uma dependência de um-para-muitos entre os objetos, de forma que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.**

Padrões de Projeto

- **Observador (Observer)**



Padrões de Projeto

- **Estratégia (Strategy)**
  - **Utilizado quando:**
    - várias classes relacionadas diferem apenas no comportamento ou
    - são necessárias diversas versões de um algoritmo ou
    - as aplicações-cliente não precisam saber detalhes específicos de estruturas de dados de cada algoritmo.

Padrões de Projeto

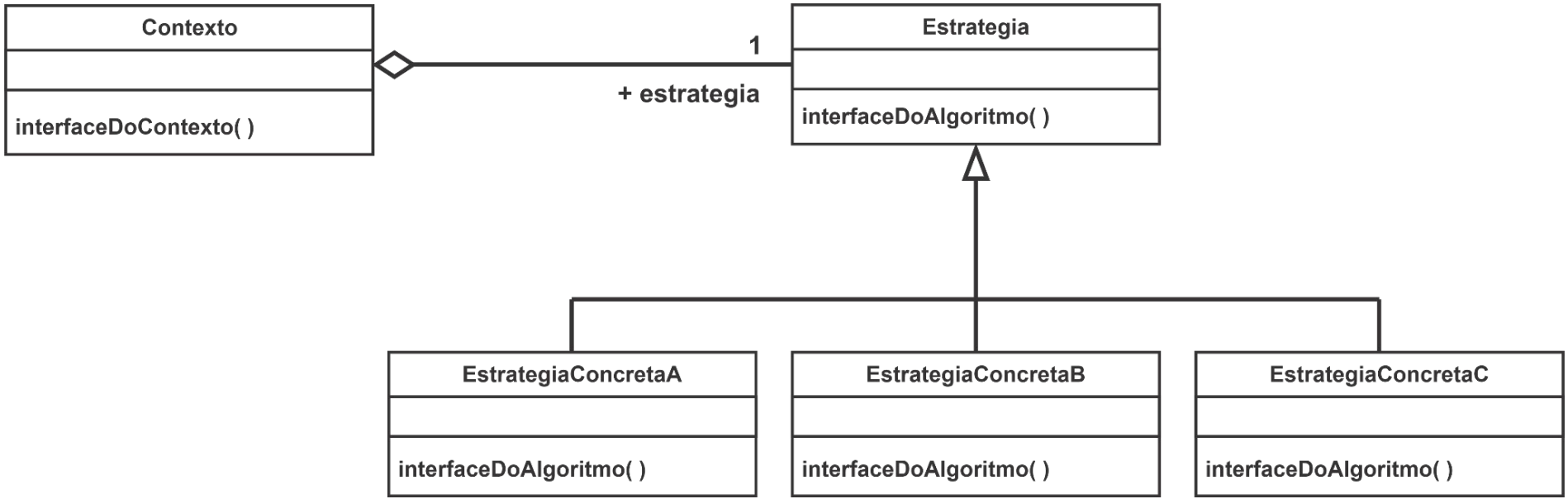
- **Estratégia (Strategy)**
  - **Problema:**
    - **Como permitir que diferentes algoritmos alternativos sejam implementados e usados em tempo de execução?**
  - **Forças:**
    - **O fato de um algoritmo diferente poder ser selecionado para realizar uma determinada tarefa, dependendo da aplicação-cliente, pode ser solucionado com uma estrutura case. Mas isso leva a projetos difíceis de manter e código redundante.**
    - **Usar herança é uma alternativa, mas também tem seus problemas: várias classes relacionadas são criadas, cuja única diferença é o algoritmo que empregam.**

Padrões de Projeto

- **Estratégia (Strategy)**
  - **Solução:**
    - **Criar uma classe abstrata para a Estratégia empregada pelo algoritmo, bem como subclasses especializando cada um dos algoritmos.**
    - **O Contexto mantém uma referência para o objeto Estratégia e pode definir uma interface para permitir que a Estratégia acesse seus dados. A Estratégia define uma interface comum a todos os algoritmos disponíveis. O Contexto delega as solicitações recebidas das aplicações-cliente para sua estratégia.**

Padrões de Projeto

- **Estratégia (Strategy)**



Padrões de Projeto

- **Quais são os padrões de projeto mais comuns?**
  - **Factory:** o padrão Factory é usado para encapsular a criação de objetos. Isso permite que você isole as dependências de criação de objetos do restante do código.
  - **Template Method:** o padrão Template Method é usado para definir o esqueleto de um algoritmo em uma superclasse, permitindo que as subclasses forneçam implementações específicas para partes do algoritmo.

Há muitos outros padrões de projeto que podem ser úteis para melhorar a qualidade interna do software. Ao aplicar esses padrões, os desenvolvedores podem tornar o software mais fácil de entender, modificar e manter, e, portanto, melhorar sua qualidade interna.

Ferramentas e tecnologias

- **Quais são as ferramentas e tecnologias mais comuns usadas para monitorar a qualidade interna do software e implementar estratégias de código limpo?**
  - **Ferramentas de análise estática de código:** como já mencionado, essas ferramentas ajudam a identificar problemas no código sem a necessidade de executá-lo. Vejam aula 10.
  - **Frameworks de teste de unidade:** frameworks como JUnit, NUnit, PHPUnit e MSTest permitem a criação e execução de testes de unidade automatizados.

Ferramentas e tecnologias

- **Quais são as ferramentas e tecnologias mais comuns usadas para monitorar a qualidade interna do software e implementar estratégias de código limpo?**
  - **Ferramentas de revisão de código:** como o próprio nome sugere, essas ferramentas ajudam a facilitar a revisão de código por parte da equipe. Exemplos incluem GitHub, Bitbucket e GitLab.
  - **Ferramentas de integração contínua:** como Jenkins, CircleCI e Travis CI, que automatizam a construção, teste e implantação contínua de software.

Ferramentas e tecnologias

- **Quais são as ferramentas e tecnologias mais comuns usadas para monitorar a qualidade interna do software e implementar estratégias de código limpo?**
  - **Ferramentas de gerenciamento de dependências:** como Maven, Gradle e npm, que ajudam a gerenciar as dependências do projeto e garantir a compatibilidade entre elas.
  - **IDEs:** ambientes integrados de desenvolvimento como Eclipse, IntelliJ e Visual Studio, que possuem recursos integrados de análise estática de código, refatoração e depuração.

Ferramentas e tecnologias

- **Quais são as vantagens e desvantagens dessas ferramentas e tecnologias?**
  - Algumas vantagens dessas ferramentas e tecnologias incluem a automação de tarefas tediosas, a detecção precoce de problemas e a melhoria da colaboração e comunicação da equipe.
  - No entanto, é importante lembrar que essas ferramentas não substituem a habilidade e julgamento humano e que é preciso avaliar cuidadosamente as vantagens e desvantagens de cada uma antes de decidir utilizá-las. Além disso, é importante considerar que essas ferramentas podem exigir um certo investimento financeiro e de tempo para serem implementadas e configuradas corretamente.