# Análise Orientada a Objetos

Design orientado a objetos

Softblue
cursos online

1

## Tópicos Abordados

- Iniciando o design
- Diagrama de classes
  - Estudo de caso: *AirBlue*
- Diagrama de sequência
  - Estudo de caso: *AirBlue*
- Criação de diagramas UML
- Princípios GRASP
- Visibilidade
- Encapsulamento

2

## Iniciando o Design

- O design inicia depois que a fase de análise é concluída
- Antes de iniciar o design, alguns detalhes já devem estar claros
  - Linguagem de programação
  - Frameworks e bibliotecas
  - Sistema operacional
  - Restrições técnicas
- O design foca nos aspectos de implementação da solução final

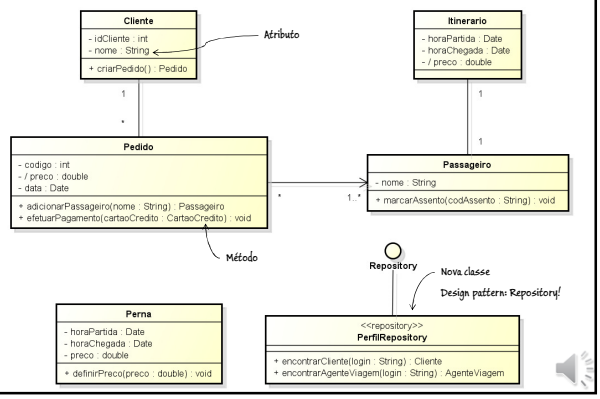
3

## Diagrama de Classes

- O diagrama de classes que representa o modelo do domínio ganha mais detalhes e passa a representar as classes que serão realmente implementadas
  - As classes conceituais são mapeadas para as classes a serem implementadas
  - Novas classes podem surgir
  - As classes passam a conter métodos (operações)

4

## *AirBlue*: Diagrama de Classes



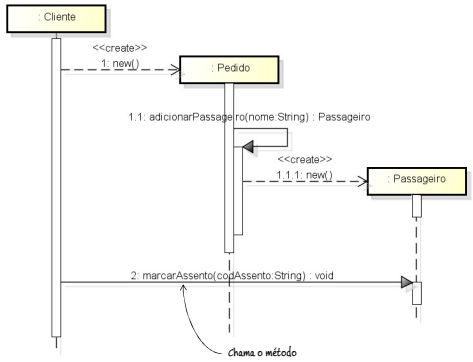

5

## Diagrama de Sequência

- Deixa de ser um diagrama de sequência de sistema
- Passa a mostrar a sequência de chamadas a métodos existentes em objetos

6

## *AirBlue*: Diagrama de Sequência




7

## Criação de Diagramas UML

- Diagramas de classe e sequência são os mais utilizados neste ponto do projeto
  - O diagrama de classes fornece uma visão estática
  - O diagrama de sequência fornece uma visão dinâmica
- Não é necessário fazer uma representação detalhada sobre tudo
  - A ideia é trabalhar iterativamente, evoluindo o design e o código

8

## Princípios GRASP

- **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**rinciples
- São um guia para definir as responsabilidades dos objetos durante a fase de design
- Cada princípio é na verdade um *pattern* (padrão)
  - Possui um nome
  - Define um problema a ser resolvido
  - Define uma solução a ser adotada
- 9 princípios fazem parte do GRASP

9

## GRASP: *Controller*

- Problema
  - Quando o usuário executar uma ação na interface gráfica, quem vai ser responsável por detectar esta ação e agir?
- Solução
  - A responsabilidade deve ser de uma classe geral para aplicação toda
    - Ex: *GeneralController*
  - A responsabilidade deve ser uma classe associada a um caso de uso específico
    - Ex: *PerfilController*, *ViagemHandler*

10

## GRASP: *Creator*

- Problema
  - De quem é a responsabilidade de criar uma nova instância de uma classe?
- Solução
  - A classe *B* deve ser responsável por criar objetos da classe *A* se:
    - *B* possui os dados de inicialização de *A*, que deverão ser passados a *A* quando este for criado
    - *B* contém *A* ou *A* é parte de *B*
    - *B* tem uma relação muito forte com *A*
    - *B* armazena *A*

11

## GRASP: *High Cohesion*

- Alta coesão
  - A coesão mede a força e o foco das responsabilidades em um elemento
- Problema
  - Como manter os objetos focados em uma tarefa e fáceis de entender e também manter?
- Solução
  - Atribua responsabilidades de forma a manter a alta coesão
  - Um objeto não deve fazer várias tarefas não relacionadas
  - Um objeto deve centralizar tarefas relacionadas

12

## GRASP: *Low Coupling*

Softblue

- Baixo acoplamento
  - O acoplamento mede o nível de dependência e relação de um elemento com outros
- Problema
  - Como manter um baixo grau de dependência, a fim de favorecer a reutilização de código e diminuir o impacto causado por mudanças?
- Solução
  - Atribua responsabilidades de forma a manter o baixo acoplamento
  - Busque reduzir o grau de dependência entre objetos

13

## GRASP: *Information Expert*

Softblue

- Problema
  - Qual princípio aplicar para determinar para qual objeto uma responsabilidade deve ser delegada?
- Solução
  - Determine quais informações são necessárias pela tarefa
  - Delegue a responsabilidade ao objeto que tem o maior número de informações para completar esta tarefa

14

## GRASP: *Indirection*

Softblue

- Problema
  - Onde colocar uma responsabilidade a fim de evitar o acoplamento direto entre dois ou mais elementos?
  - Como desacoplar objetos para aumentar o grau de reuso?
- Solução
  - Delegue a responsabilidade a um objeto intermediário
  - Este objeto vai mediar o acesso entre os elementos, que não estarão diretamente acoplados

15

## GRASP: *Pure Fabrication*

Softblue

- Problema
  - Qual objeto deve receber determinada responsabilidade a fim de manter a alta coesão e o baixo acoplamento quando o princípio *information expert* não for adequado?
- Solução
  - Crie uma classe que não tem relação com o modelo do domínio e delegue a ela esta responsabilidade

16

## GRASP: *Protected Variations*

Softblue

- Problema
  - Como fazer o design de subsistemas de forma que a instabilidade nesse design não afete outros elementos da solução?
- Solução
  - Identifique os pontos onde esta instabilidade possa ocorrer
  - Crie uma interface bem definida para "esconder" esta instabilidade

17

## GRASP: *Polymorphism*

Softblue

- Problema
  - Como criar alternativas baseadas em um tipo?
  - Como criar componentes de software plugáveis?
- Solução
  - Delegue as responsabilidades para os tipos polimórficos (tipos onde o comportamento varia)

18

## Visibilidade

- A visibilidade determina se um elemento pode "ver" outro
  - Classes, atributos, métodos, pacotes
- Defina a visibilidade de acordo com a necessidade
  - Linguagens de programação costumam permitir vários tipos de visibilidade
    - **Privada**: visível só para o próprio objeto
    - **Pública**: visível para todos
    - **Protegida**: visível às subclasses
    - **Pacote**: visível dentro de um pacote

19

## Encapsulamento

- O encapsulamento é um princípio importante na orientação a objetos
- Determina que detalhes de implementação de um objeto não devem ser expostos para fora deste objeto
  - Atributos devem ser definidos privados
  - A manipulação dos atributos deve ser feita apenas através de métodos, criados com esta finalidade

20


21