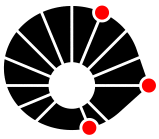

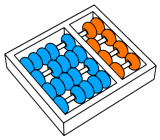

Grupo Caramelo: Sistema de gerenciamento de academia

Ana Clara Zoppi Serpa - RA 165880
Bruno de Marco Apolonio - RA 195036
Gabriel Oliveira dos Santos - RA 197460
Lucas Costa de Oliveira - RA 182410
Vítor Mosso Dario - RA 207024

Technical Report - IC-18-01 - Relatório Técnico
June - 2018 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Caramelo: Sistema de gerenciamento de academia

Ana Clara Zoppi Serpa Bruno de Marco Apolonio Gabriel Oliveira dos Santos
Lucas Costa de Oliveira Vítor Mosso Dario

Resumo

Quisque accumsan ipsum id tortor laoreet feugiat. In accumsan id risus quis rutrum. Aliquam risus nunc, lacinia ac tincidunt at, accumsan ut purus. Morbi vestibulum et lacus at interdum. Cras pellentesque consectetur sapien, ac lobortis leo aliquam at. In consectetur nibh at bibendum laoreet. Aenean molestie lorem id mattis mattis. Integer rhoncus sem dictum, mattis massa vitae, pretium justo. Curabitur euismod dolor non neque semper tincidunt.

In congue consectetur risus eu pharetra. Vestibulum at lacus auctor, cursus leo et, placerat nibh. Suspendisse vitae enim justo. Praesent feugiat dolor accumsan dui euismod blandit. Maecenas interdum velit dolor, in laoreet urna feugiat finibus. Ut cursus eros id velit laoreet, at fringilla nibh ullamcorper. Donec aliquet elit nisi, quis pellentesque nibh molestie quis. Praesent volutpat hendrerit augue id molestie. Nullam orci ex, auctor non tristique vitae, dignissim at mauris. Fusce tempor eleifend aliquet. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Proin mollis ligula id laoreet dignissim.

Sumário

1 Etapa 0: Proposta de Software

Nessa etapa, decidimos qual seria o tema do nosso projeto e quais suas funcionalidades. Optamos por implementar um sistema de gerenciamento de academias com cadastro de clientes, atividades e instrutores e a associação destes, por exemplo: o cliente Gabriel faz pilates de segunda e quarta, das 16h às 17h, e o instrutor responsável pelas aulas de pilates desse horário é o Lucas.

Escrevemos um documento descrevendo as funcionalidades que previámos para o sistema e entregamos para o professor. Como elas já foram descritas no documento, aqui são mencionadas mais brevemente:

- Cadastro dos clientes que frequentam a academia
- Alteração dos dados de um cliente já cadastrado
- Desativar um cliente
- Cadastro de atividades que a academia oferece
- Alteração dos dados da atividade, por exemplo, preço
- Remoção de uma atividade
- Associar/desassociar horários às atividades oferecidas
- Associar/desassociar clientes aos horários e atividades
- Cadastrar instrutores que trabalham na academia
- Alterar dados de um instrutor cadastrado
- Desativar um instrutor
- Consultar qual atividade possui mais clientes
- Consultar qual atividade possui menos clientes
- Consultar qual atividade tem o maior preço
- Consultar qual atividade tem o menor preço

Em vez de remover clientes e instrutores, optamos por desativá-los. Tivemos essa ideia pensando no seguinte cenário: um cliente decide deixar de frequentar a academia, mas, após algum tempo, retorna. Seria prático se já tivéssemos os dados dele e bastasse reativá-lo. O mesmo ocorre com instrutores que parem de trabalhar na academia, mas depois retornem. Ou, por exemplo, a academia talvez tenha interesse em tentar contactar um instrutor para contratá-lo novamente, ou um cliente antigo para perguntar se ele deseja retornar.

2 Etapa 1: Classes do sistema

Nessa etapa, pensamos sobre as classes e relacionamentos entre elas e construímos um diagrama UML. O professor indicou correções, as quais realizamos em outras etapas. O diagrama a seguir está parcialmente correto e o apresentamos apenas para facilitar a compreensão das ideias.

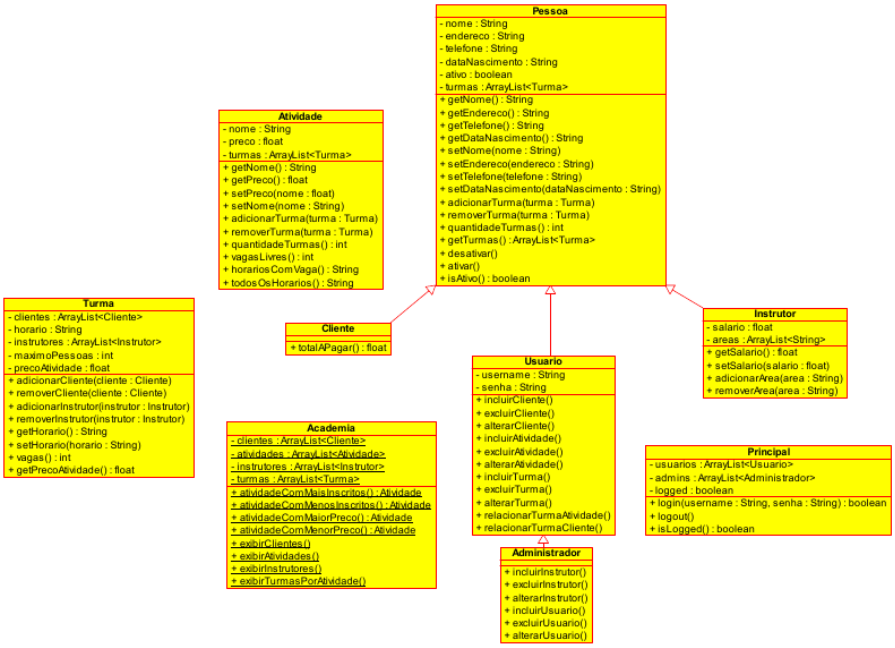

Figura 1: Diagrama UML para a Release 0

Decidimos que o relacionamento entre cliente, horário, atividade e instrutor seria feito por meio de uma nova classe, a classe Turma, da seguinte forma:

- Uma atividade possui n turmas.
- Uma turma possui n clientes, n instrutores e um horário que representaremos com o tipo String.
- Um cliente possui n turmas, afinal, ele pode realizar várias atividades diferentes.
- Um instrutor possui n turmas porque ele pode dar aulas em várias turmas diferentes.

Percebemos que clientes e instrutores possuiriam nome, endereço, telefone, data de nascimento, lista de turmas e a indicação de que estão ou não ativos no sistema como atributos em comum. Portanto, decidimos criar a classe Pessoa com esses atributos, getters e setters e fazer com que clientes e instrutores fossem suas subclasses.

O cliente possui, além dos métodos e atributos da classe Pessoa, o método que calcula quanto ele precisa pagar pelas atividades realizadas.

Como uma atividade possui n turmas, mas as turmas possuem apenas clientes, instrutores e horário, ou seja, a turma não sabe qual é a sua atividade, decidimos, por questões de facilidade, guardar o preço da atividade na classe Turma também. Dessa forma, para calcular quanto o cliente precisa pagar, basta iterar por sua lista de turmas e acumular os preços.

O instrutor possui, além dos métodos e atributos da classe Pessoa, um atributo salário e uma lista de áreas (que representamos com uma lista de Strings) nas quais atua.

A classe Academia seria nossa base de dados, guardando em listas todos os clientes, instrutores, atividades e turmas do sistema e possuindo métodos para exibí-los.

Decidimos também implementar uma etapa de login no sistema e separar os usuários entre usuários comuns e usuários administradores. Um usuário comum poderia incluir/alterar/remover/desativar clientes, atividades e turmas, enquanto um usuário administrador poderia fazê-lo também para instrutores e usuários. Portanto, administrador é uma subclasse de usuário.

Usuários possuem username e senha. Nessa etapa do projeto, pensamos em fazer com que Usuário fosse subclasse de Pessoa, para guardarmos também seus dados (nome, endereço, telefone, data de nascimento).

3 Etapa 2: Release 0

Para essa Release, iniciamos a implementação seguindo o diagrama UML construído na Etapa 1. Criamos as classes em Java, seus métodos e implementamos alguns deles, deixando outros vazios para serem implementados posteriormente.

4 Etapa 3: Release 1

Para essa Release, implementamos os métodos que, na anterior, estavam vazios. Realizamos algumas mudanças ao perceber alternativas e necessidades não previstas. Atualizamos o diagrama UML para que refletisse essas mudanças, acrescentando algumas correções mencionadas pelo professor (cardinalidades, associações, agregações etc).

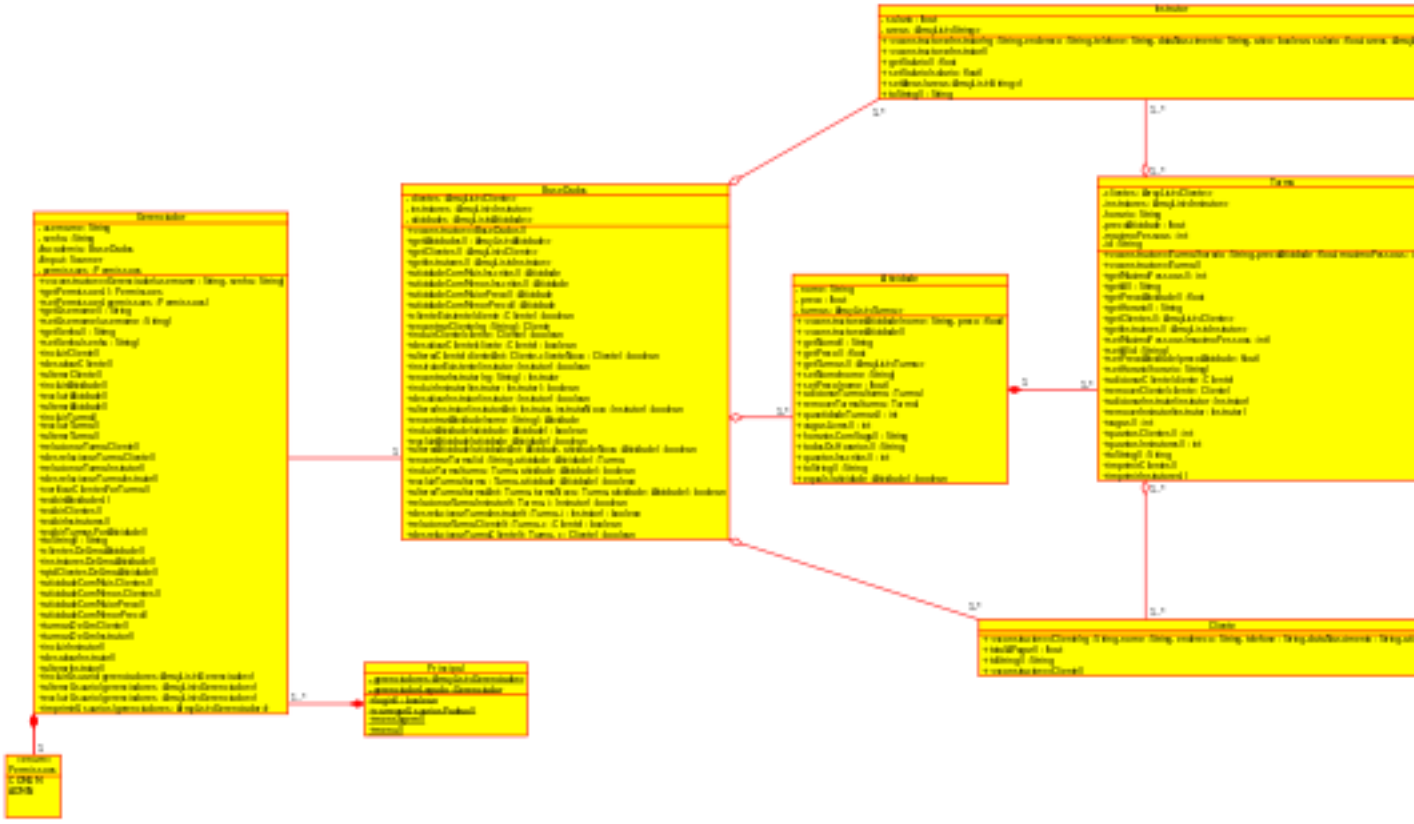

Figura 2: Diagrama UML para a Release 1

- ficou faltando herança, conceitos de POO e buscas
- fizemos protótipos de interface gráfica

5 Etapa 4: Release 2

- colocamos conceitos de POO e buscas - mais interface gráfica

6 Release final

- colocamos o login que tínhamos tirado - mais interface gráfica