Introdução

Este projeto faz parte da disciplina "Programação I" e tem como objetivo consolidar os fundamentos de programação orientada a objetos em Java, utilizando o ecossistema do JavaFX para construção de interfaces gráficas. A aplicação simula uma fruteira, permitindo visualizar produtos, adicionar itens ao carrinho, confirmar pedidos e realizar operações básicas de administração, tudo organizado em camadas (models, controllers e views) para reforçar boas práticas de projeto.

Objetivos do Projeto

- Desenvolver uma aplicação Java utilizando JavaFX, praticando a criação de interfaces gráficas e a navegação entre telas.

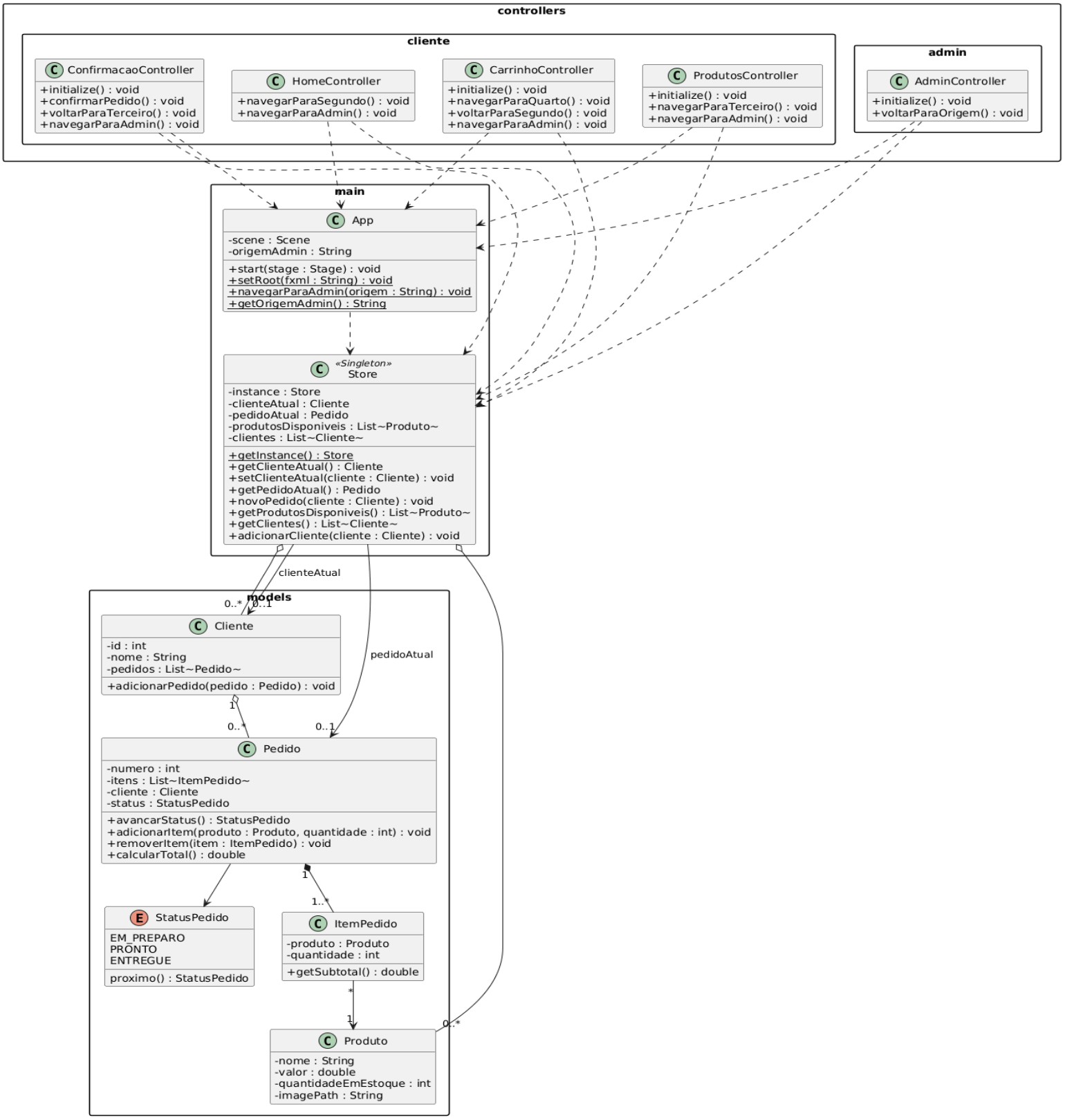
- Aplicar conceitos de Programação Orientada a Objetos (POO): classes, encapsulamento, composição e enums, por meio dos modelos Produto, Cliente, Pedido, ItemPedido e StatusPedido.

- Estruturar o código seguindo o padrão MVC simplificado (models, controllers, fxml/views), melhorando separação de responsabilidades e manutenção.

- Implementar funcionalidades típicas de um e-commerce simples: listagem de produtos, carrinho de compras, confirmação de pedido e área administrativa.

- Utilizar ferramentas de build e testes (Maven e JUnit) para compilar, executar e validar o comportamento da aplicação.

Diagrama de Classes UML



Principais Componentes

- App (entrada da aplicação): inicializa o JavaFX, carrega as primeiras views (home.fxml) e configura o contexto necessário.

- Store (serviços e estado): atua como uma "fachada" simples que centraliza operações do domínio (catálogo, carrinho, pedidos) e fornece métodos usados pelos controllers.

Controllers (camada de controle):

- HomeController, ProdutosController, CarrinhoController, ConfirmacaoController (cliente): coordenam interação entre a UI e os modelos, acionando a Store.

- AdminController (admin): operações administrativas como cadastro/atualização de produtos.

Models (camada de domínio):

- Produto, Cliente, Pedido, ItemPedido, StatusPedido (enum): representam entidades e regras básicas do negócio.

- Views/Recursos (UI): arquivos FXML (home.fxml, produtos.fxml, carrinho.fxml, confirmacao.fxml, admin.fxml), estilos (styles.css) e imagens dos produtos.

- Testes: classes JUnit em src/test/java (StoreTest, PedidoTest, StatusPedidoTest) validam regras essenciais do domínio.

Decisões de Modelagem

- Padrão MVC simplificado: separação entre Models (lógica e dados), Controllers (orquestração e ligação com UI) e Views (FXML/CSS). Facilita manutenção e testes.

- Enum para estados de pedido: StatusPedido garante valores controlados (ex.: PENDENTE, CONFIRMADO), reduzindo erros de strings mágicas e melhorando legibilidade.

- Composição em Pedido e ItemPedido: um Pedido agrega ItemPedido (produto + quantidade + preço) para refletir a relação natural do carrinho. Evita acoplamento excessivo entre UI e domínio.

- Imutabilidade seletiva: valores como preço unitário do Produto e identificadores tendem a ser estáveis, alterações passam por métodos específicos na Store ou em controllers, mantendo invariantes.

- Coleções internas e encapsulamento: uso de listas/mapas para catálogo e carrinho é mediado pela Store, evitando exposição direta de estruturas mutáveis e simplificando validação.

- Recursos externos organizados: imagens e arquivos FXML ficam em src/main/resources/... para facilitar empacotamento via Maven e compatibilidade com JavaFX.

- Testes de unidade focados no domínio: regras como cálculo de total do pedido e transições de StatusPedido são cobertas por testes, incentivando TDD leve e segurança em refatorações.

- Módulos e module-info.java: mantém compatibilidade com JPMS, declarando dependências e melhorando segurança/encapsulamento onde possível.