Adapter

Foi utilizado o **Adapter** no contexto de notificações. Temos duas Formas de enviar notificação: Firebase(em RAM) e Database, nós adaptamos a chamada de Notificacao com base no Adapter que pode ser DatabaseAdapterNotificatorApi ou FirebaseAdapterNotificatorApi, e com base nisso converter um objeto entidade Notificacao em uma linha do banco de dados se for Database ou um item numa array se for Firebase. Foi criado também um Singleton pois precisa-se de uma classe para enviar e receber notificacoes ao longo do programa inteiro, então ele é instanciado apenas uma vez na fachada e usado quando um administrador faz uma edição em uma loja, enviando notificaçao, ou quando um gerente ou vendedor decide ver as notificações da sua Loja, listando as notificações filtrando pelo id da loja.

Lista de Classes:

- NotificatonError

- NotificationApi

- DatabaseAdapterNotificatorApi

- FirebaseAdapterNotificatorApi

- NotificatorApiSingleton

Singleton

Foi utilizado o **Singleton** no contexto d
repositório tenha apenas uma instância
e controle centralizado do acesso aos d

Lista de Classes:

- RepositorioUsuariosRAM

- RepositorioUsuariosDB

- RepositorioLojasRAM

- RepositorioLojasDB

- RepositorioProdutosRAM

- RepositorioProdutosDB

- RepositorioVendaRAM

- RepositorioVendaDB

os repositórios. Assím, asseguramos que cada
ativa em todo o sistema, mantendo consistência
dados.

Facade

Foi utilizado o **Facade** que gerencia a lógica no fluxo das telas, proporcio
interface simplificada para realizar todas as operações e manipulações n
sistema, abstraindo a complexidade de suas interações internas.

Classes:

- Fachada

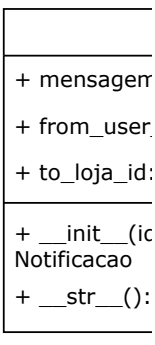
Strategy

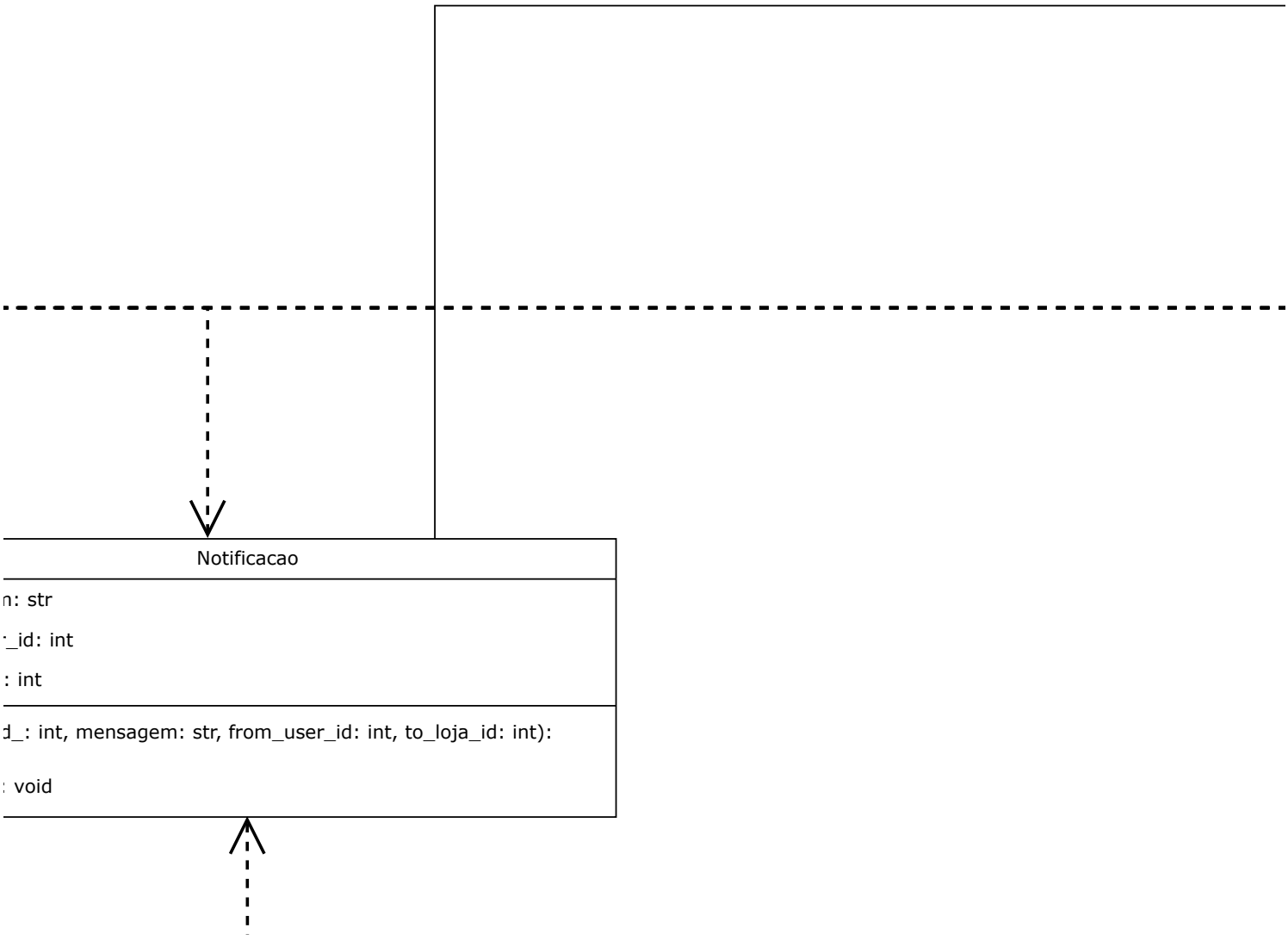
- Centralizar as operações de persistência, facilitando a troca entre difere
armazenamento para Usuários, Lojas e Produtos conforme a necessidad

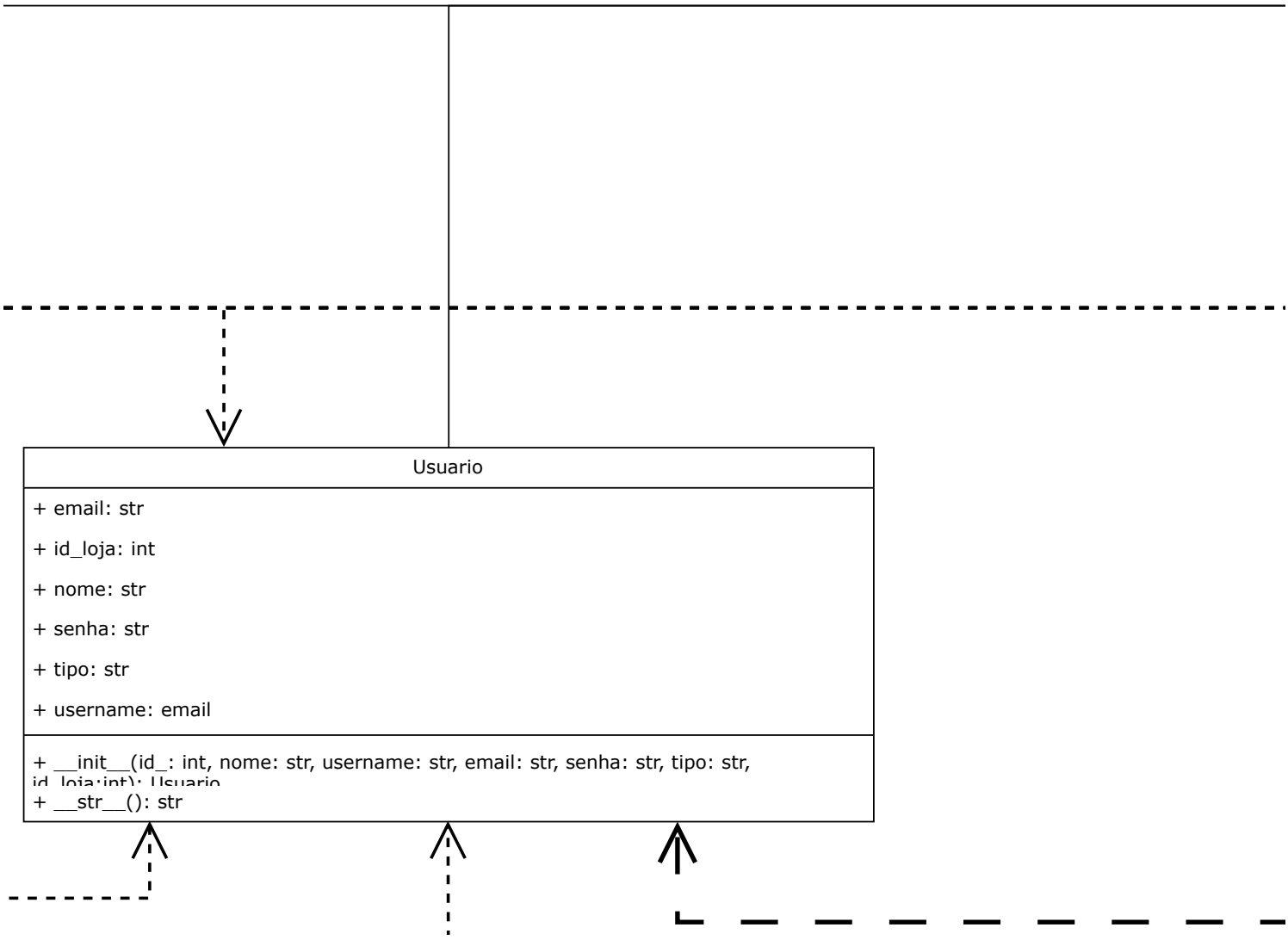
Interfaces:

onando uma
necessárias no

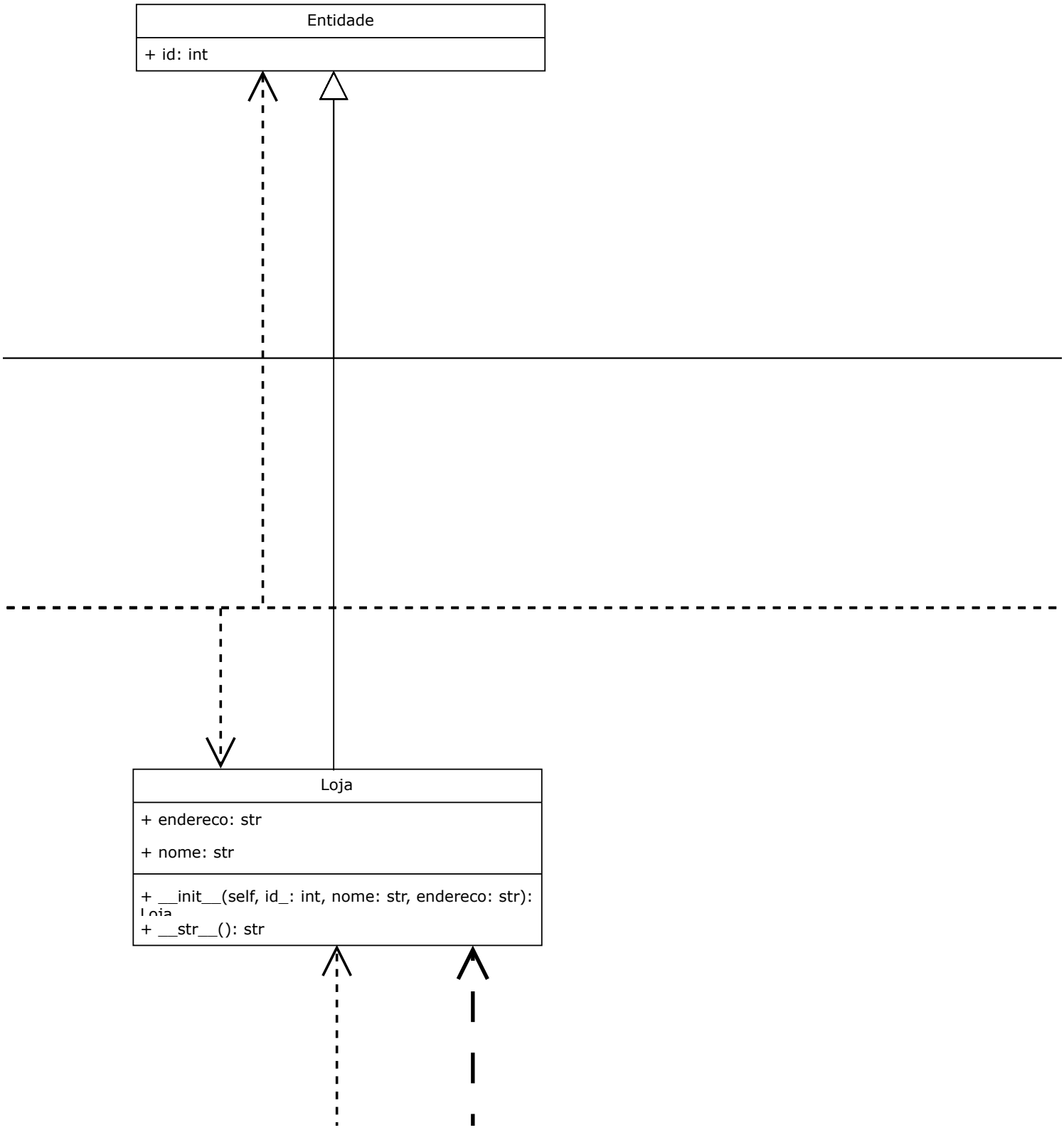
entes estratégias de
de do sistema.


+ mensagem
+ from_user
+ to_loja_id:
+ __init__(id
Notificacao
+ __str__():

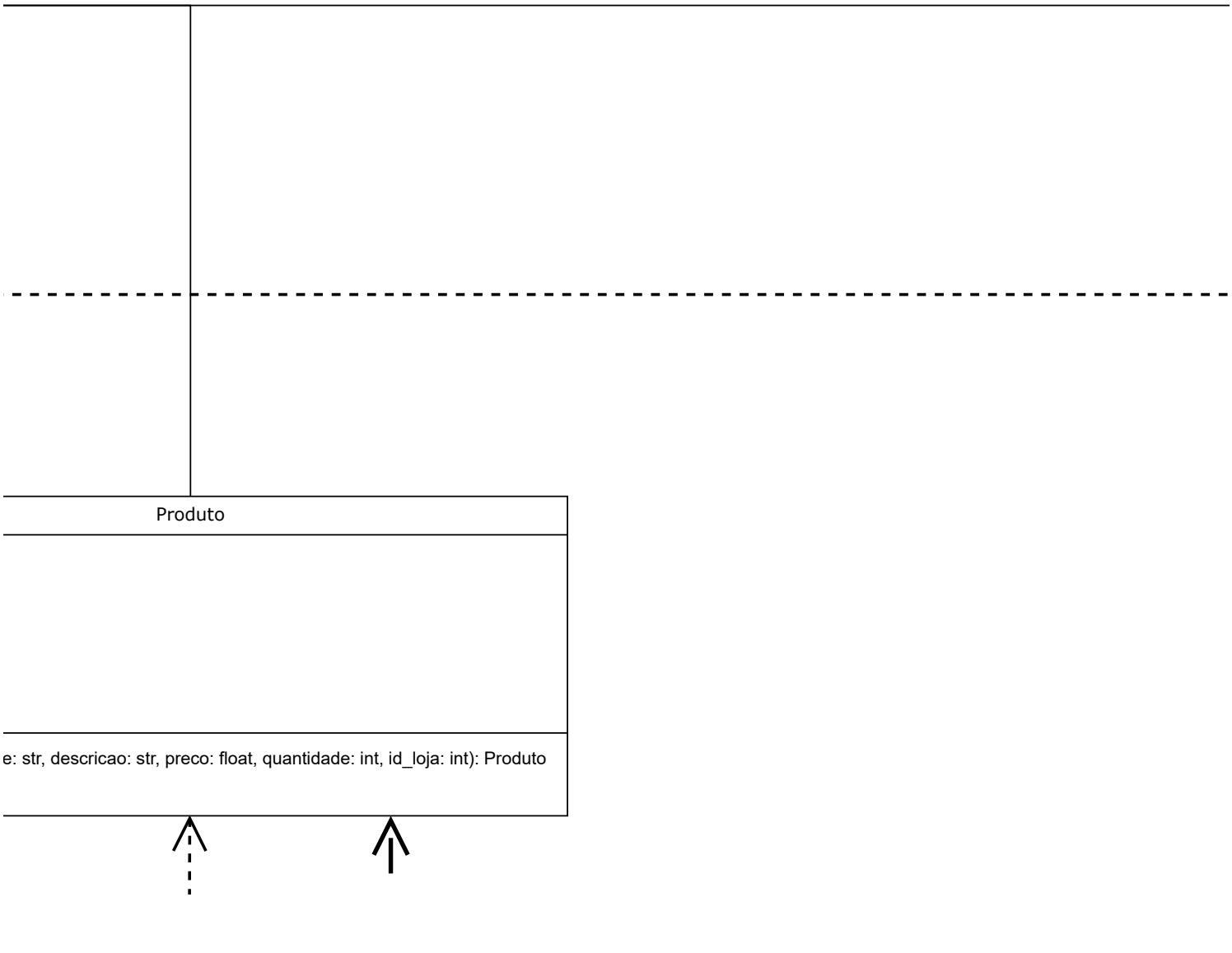

Notificacao
n: str
_id: int
: int
d_: int, mensagem: str, from_user_id: int, to_loja_id: int):
: void

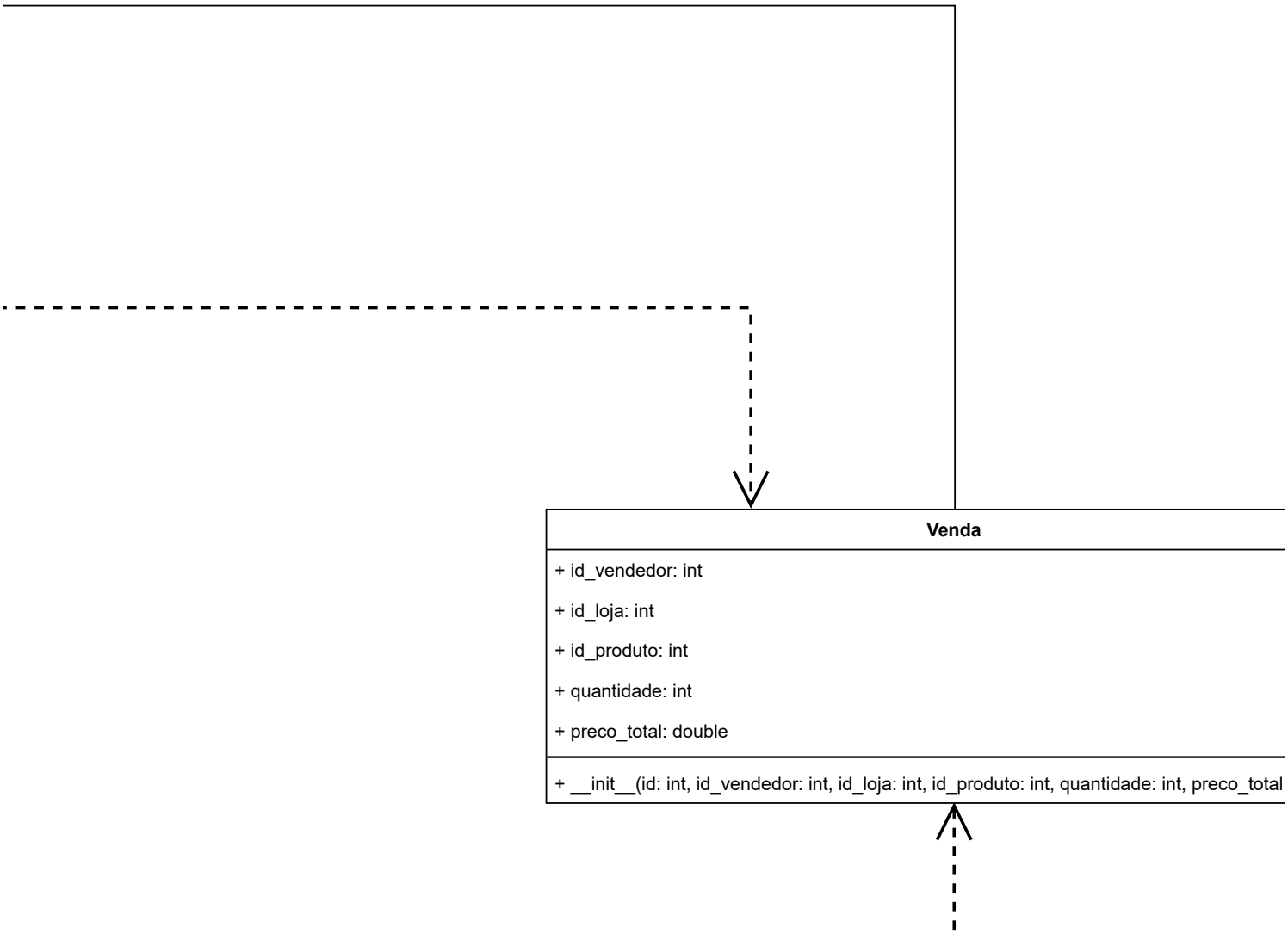

Usuario
+ email: str
+ id_loja: int
+ nome: str
+ senha: str
+ tipo: str
+ username: email
+ __init__(id_: int, nome: str, username: str, email: str, senha: str, tipo: str,
id_loja:int): Usuario
+ __str__(): str

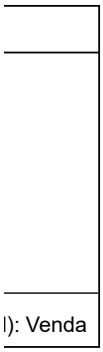
Relacionamento entre entidades




+ id_loja: int
+ nome: str
+ preco: double
+ quantidade: int
+ tipo: str
+ __init__(id_: int, nome
+ __str__(): str


Produto
e: str, descricao: str, preco: float, quantidade: int, id_loja: int): Produto


Venda
+ id_vendedor: int
+ id_loja: int
+ id_produto: int
+ quantidade: int
+ preco_total: double
+ __init__(id: int, id_vendedor: int, id_loja: int, id_produto: int, quantidade: int, preco_total


): Venda

Camada de Modelo

Memento

Foi utilizado o design pattern Memento para o contexto de editar uma loja. Primeiro instanciamos o caretaker, originator e adcionamos um state para cada modificação feita. Se o usuário decidir voltar atrás, usamos o metodo restore do originator passando o primeiro memento da lista. Usamos classes abstratas sendo implementadas pelas respectivas classes dos contextos específicos, visto que no futuro poderia-se implementar também o memento para Usuário e Produto.

Lista de Classes:

- Originator

- Memento

- Caretaker

- OriginatorLoja

- CaretakerLoja

- MementoLoja

Template Method (

Foi utilizado o design pattern **Template** usuários como administrador. O adminis formatos: HTML ou PDF, para isso criou concretos em comum quando se trata d para formatar o relatório e mostrar o su implementados por duas Classes: Relat

Lista de Classes:

- RelatórioPDF

- RelatorioHTML

- Relatorio

Template Method (Telas)

Foi utilizado o **Template Method** para a personalização de cada tela individualmente, garantindo que todas sigam um padrão estruturado definido pela classe base

Classes:

TemplateTela

Todas as Classes que representam as Telas.

Command

O objetivo principal no projeto é o de separar a interface gráfica que contém as telas, da interface de lógica. Dessa maneira, a parte da interface gráfica fica responsável por capturar os dados necessários a um comando, reuni-los, criar um comando e encaminhar para que a interface de lógica possa processar e retornar os resultados.

- Lista de Classes:

- Comando <interface dos comandos>;

- Invocador <invoker dos comandos>

- GerenciadorComandos;

- AdicionarLojaComando;

- AdicionarProdutoComando;

- AdicionarUsuarioComando;

- EditarLojaComando;

- EditarProdutoComando;

- EditarUsuarioComando;

- EnviarNotificacaoComando;

(Relatórios)

Method para a funcionalidade de criar relatórios dos
strador tem a opção de gerar relatórios em dois
u-se inicialmente uma classe abstrata com metodos
da geração desses relatórios e dois metodos abstratos
cesso da geração do relátorio. Ambos os métodos são
torioPDF e RelatorioHTML.

- InterfaceEstrategiaUsuarios
- InterfaceEstrategiaProdutos
- InterfaceEstrategiaLojas
- InterfaceEstrategiaVendas

Classes:

- EstrategiaUsuariosRAM
- EstrategiaUsuariosDB
- EstrategiaLojasRAM
- EstrategiaLojasDB
- EstrategiaProdutosRAM
- EstrategiaProdutosDB
- EstrategiaVendasRAM

* EstrategiaVendasDB

Factory Method

Foi utilizado o **Factory Method** para centralizar o processo de criação de dos gerenciadores, delegando à
responsabilidade de instanciar objetos conforme o tipo necessário.

FabricaEntidades

FabricaGerenciadorUsuarios

FabricaGerenciadorLojas

FabricaGerenciadorProdutos

FabricaGerenciadorVendas

FabricaGerenciadorNotificacoes

Abstract Factory

A Abstract Factory é uma interface que padroniza a criação de fábricas, garantindo que cada fábrica implem
método abstrato criar, que define a lógica de criação específica de cada tipo de objeto.

FabricaEntidades

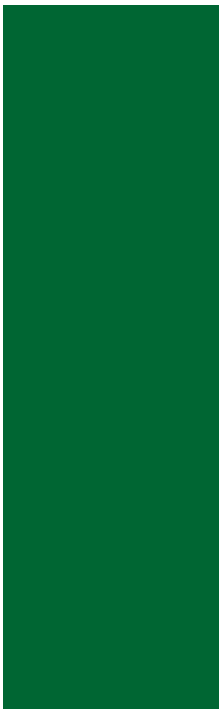
FabricaGerenciadorUsuarios

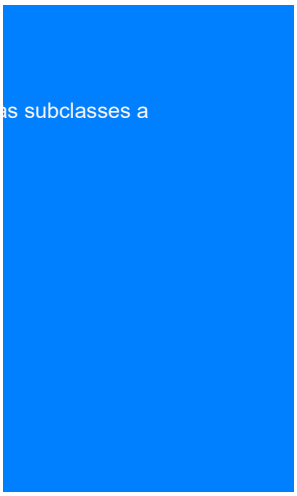
FabricaGerenciadorLojas

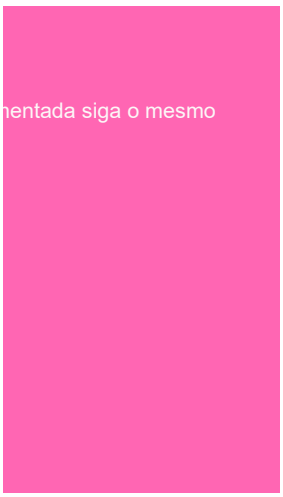
FabricaGerenciadorProdutos

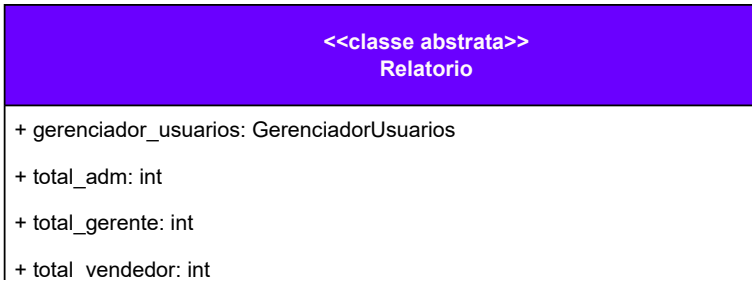
FabricaGerenciadorVendas

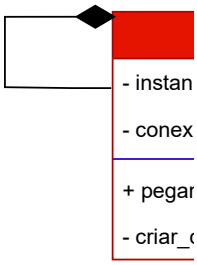
FabricaGerenciadorNotificacoes

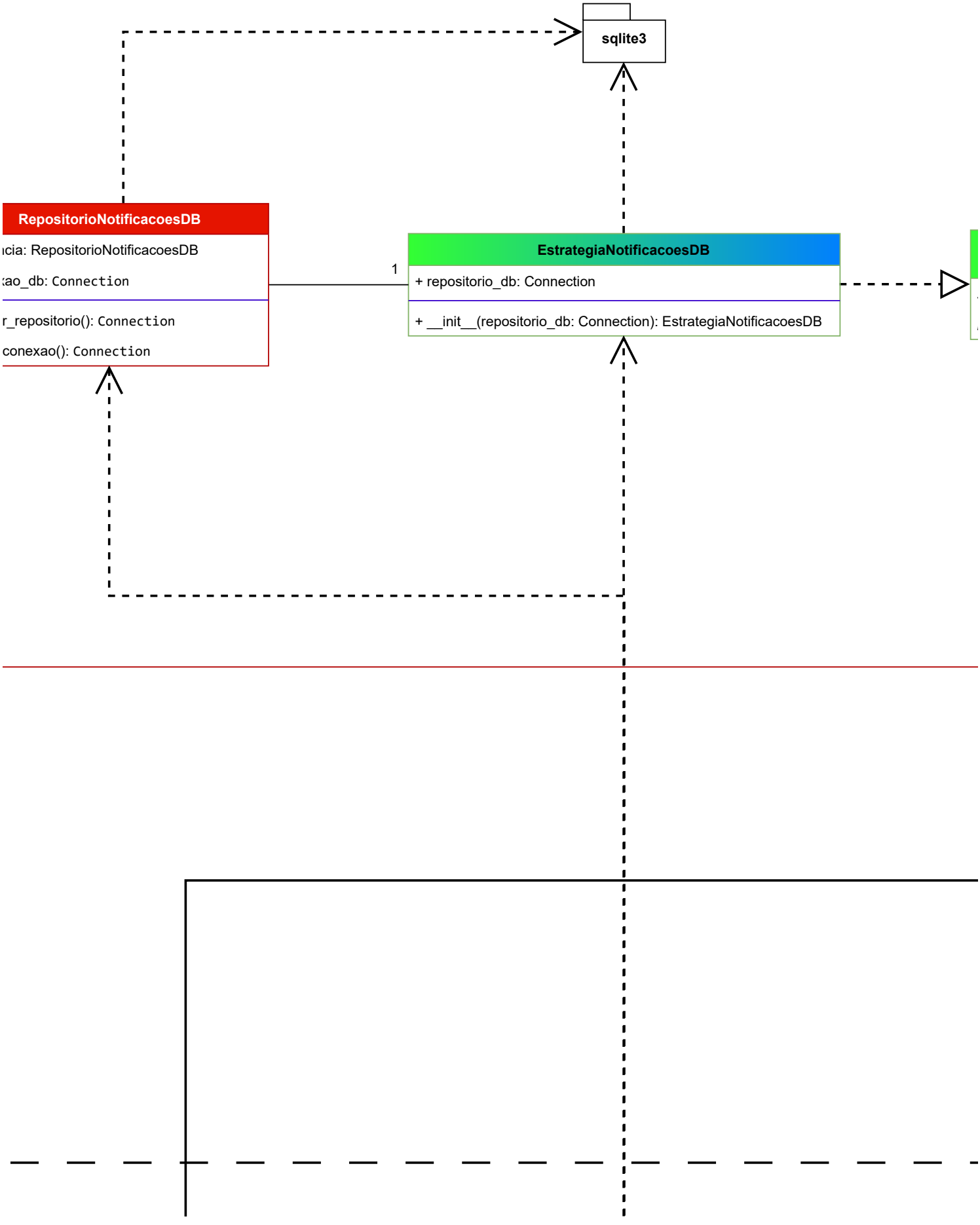


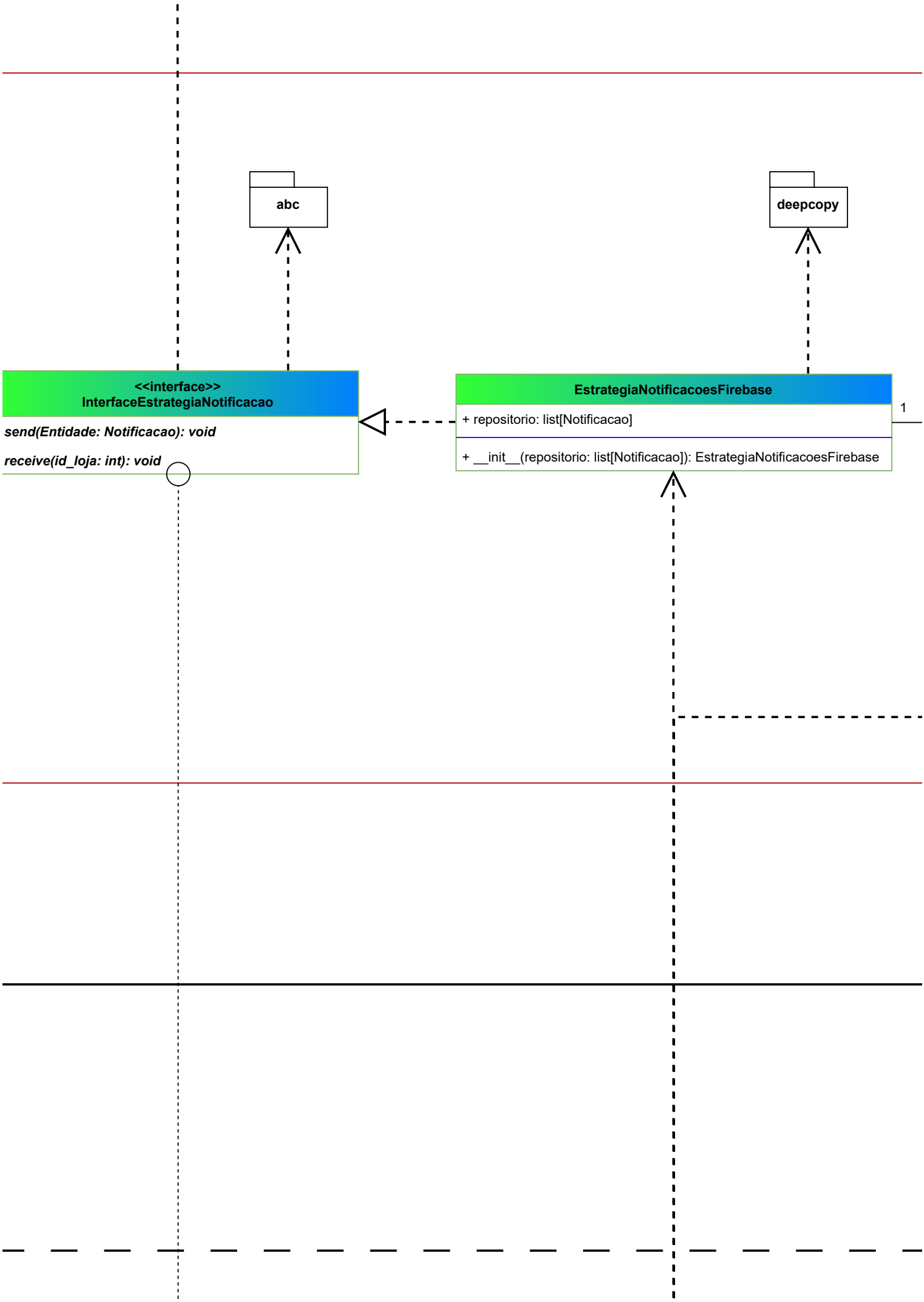

as subclasses a

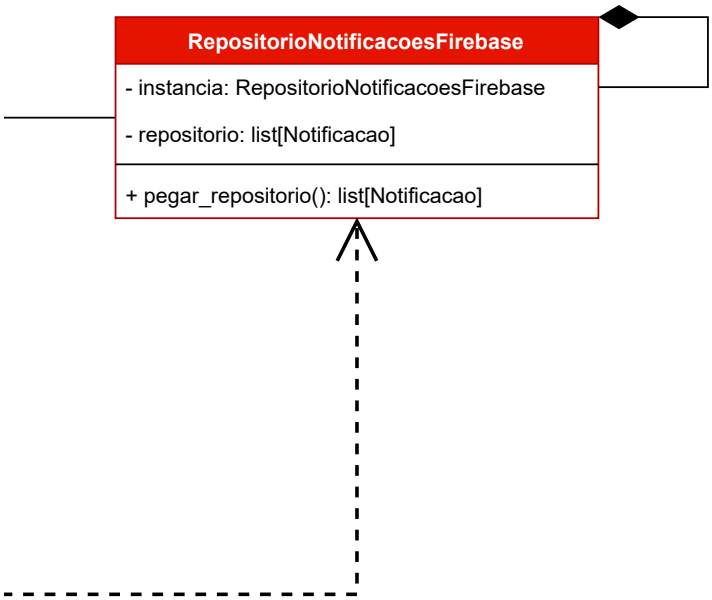

nentada siga o mesmo

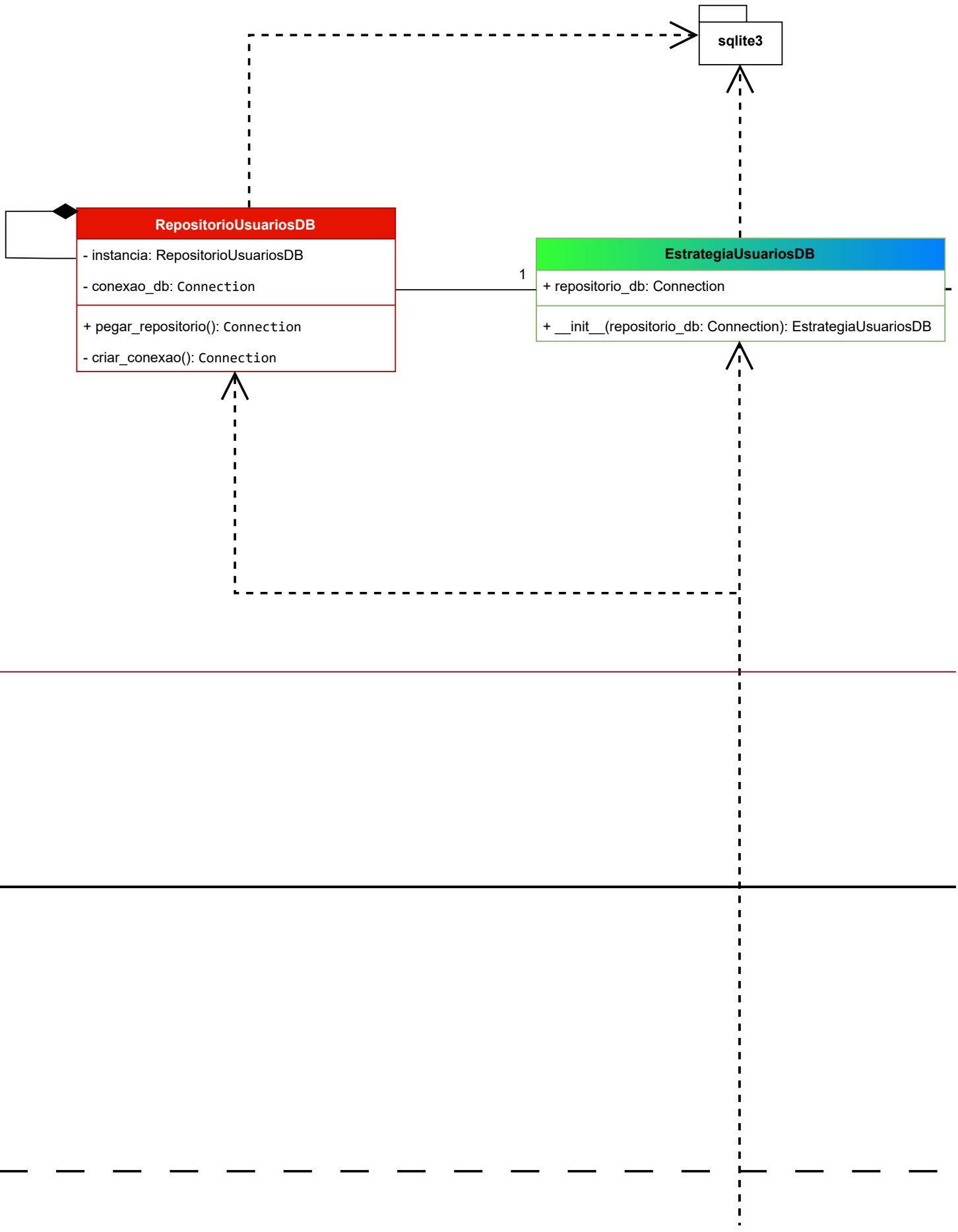

<<classe abstrata>>
Relatorio
+ gerenciador_usuarios: GerenciadorUsuarios
+ total_adm: int
+ total_gerente: int
+ total vendedor: int

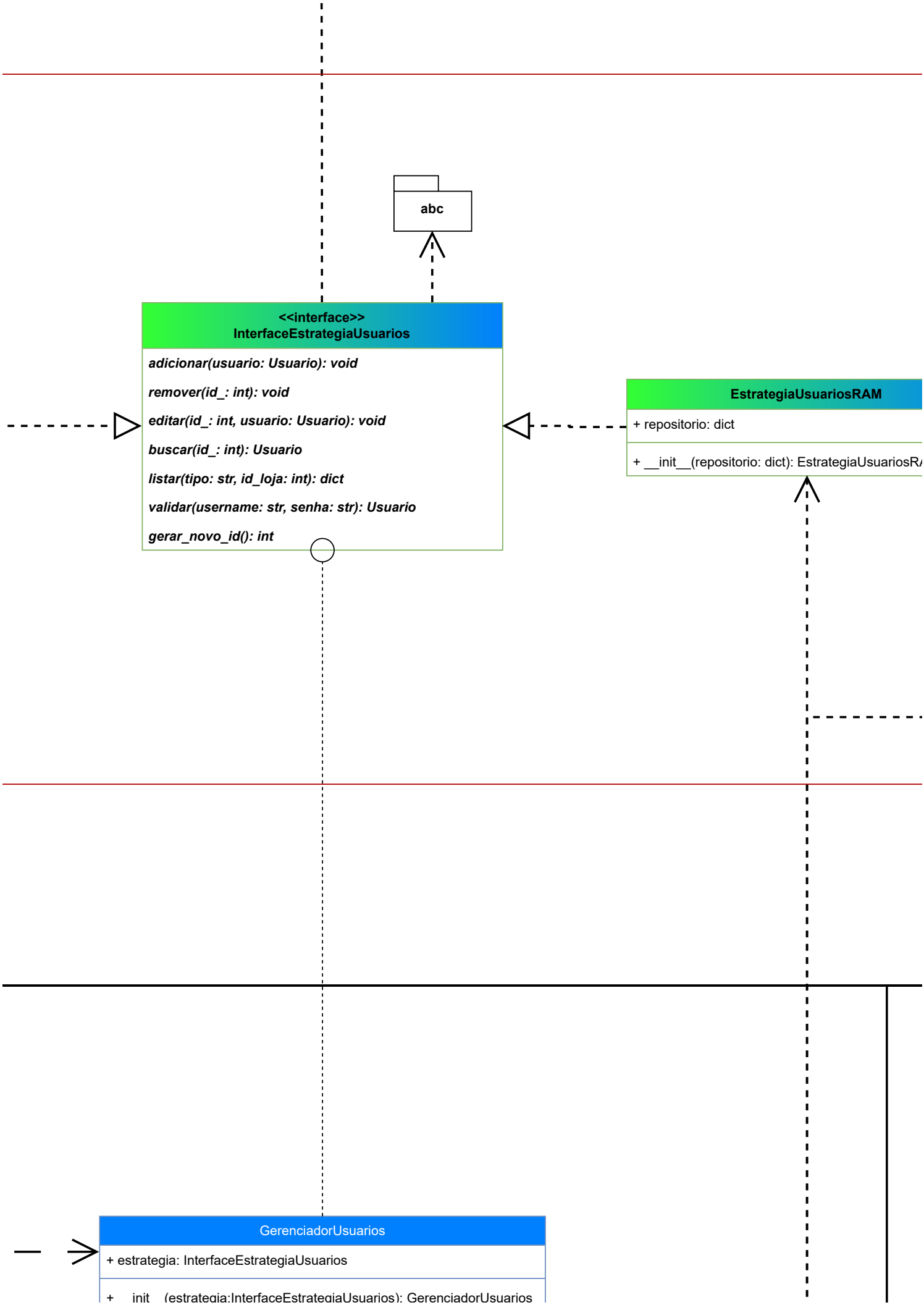

- instan
- conex
+ pegar
- criar_

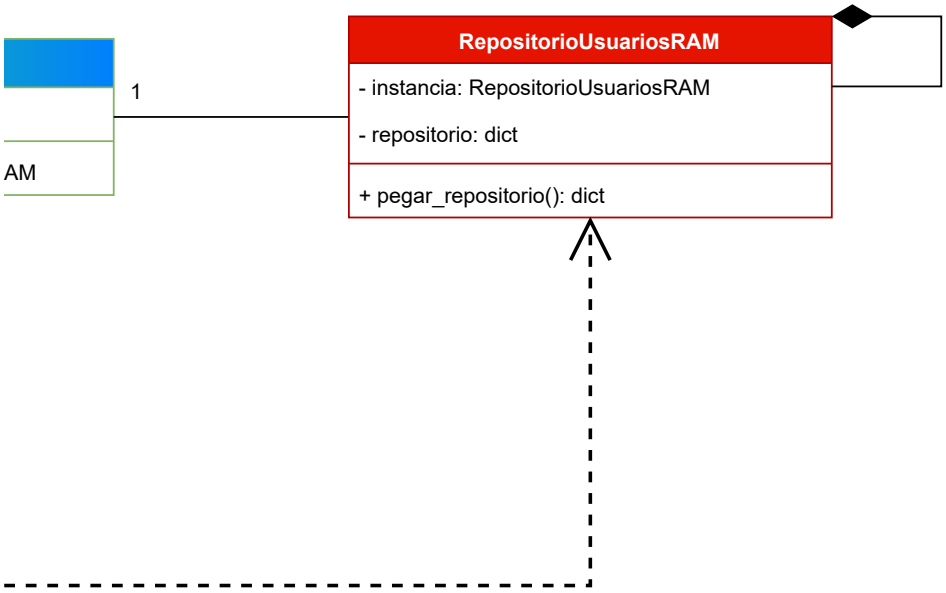

sqlite3
RepositorioNotificacoesDB
ıcia: RepositorioNotificacoesDB
:ao_db: Connection
r_repositorio(): Connection
conexao(): Connection
1
EstrategiaNotificacoesDB
+ repositorio_db: Connection
+ __init__(repositorio_db: Connection): EstrategiaNotificacoesDB

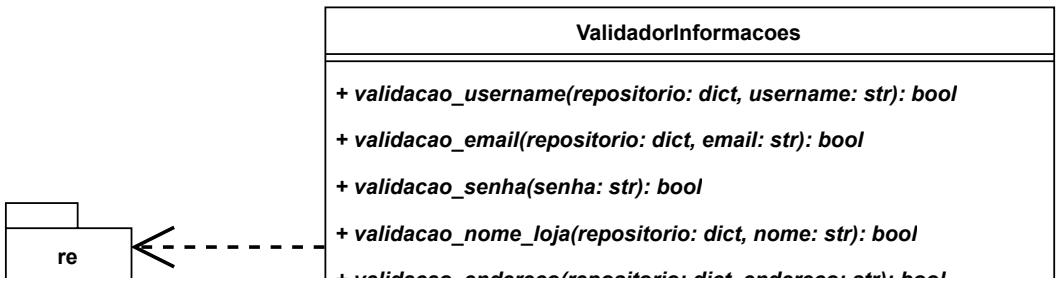

abc
deepcopy
<<interface>>
InterfaceEstrategiaNotificacao
send(Entidade: Notificacao): void
receive(id_loja: int): void
EstrategiaNotificacoesFirebase
+ repositorio: list[Notificacao]
+ __init__(repositorio: list[Notificacao]): EstrategiaNotificacoesFirebase
1

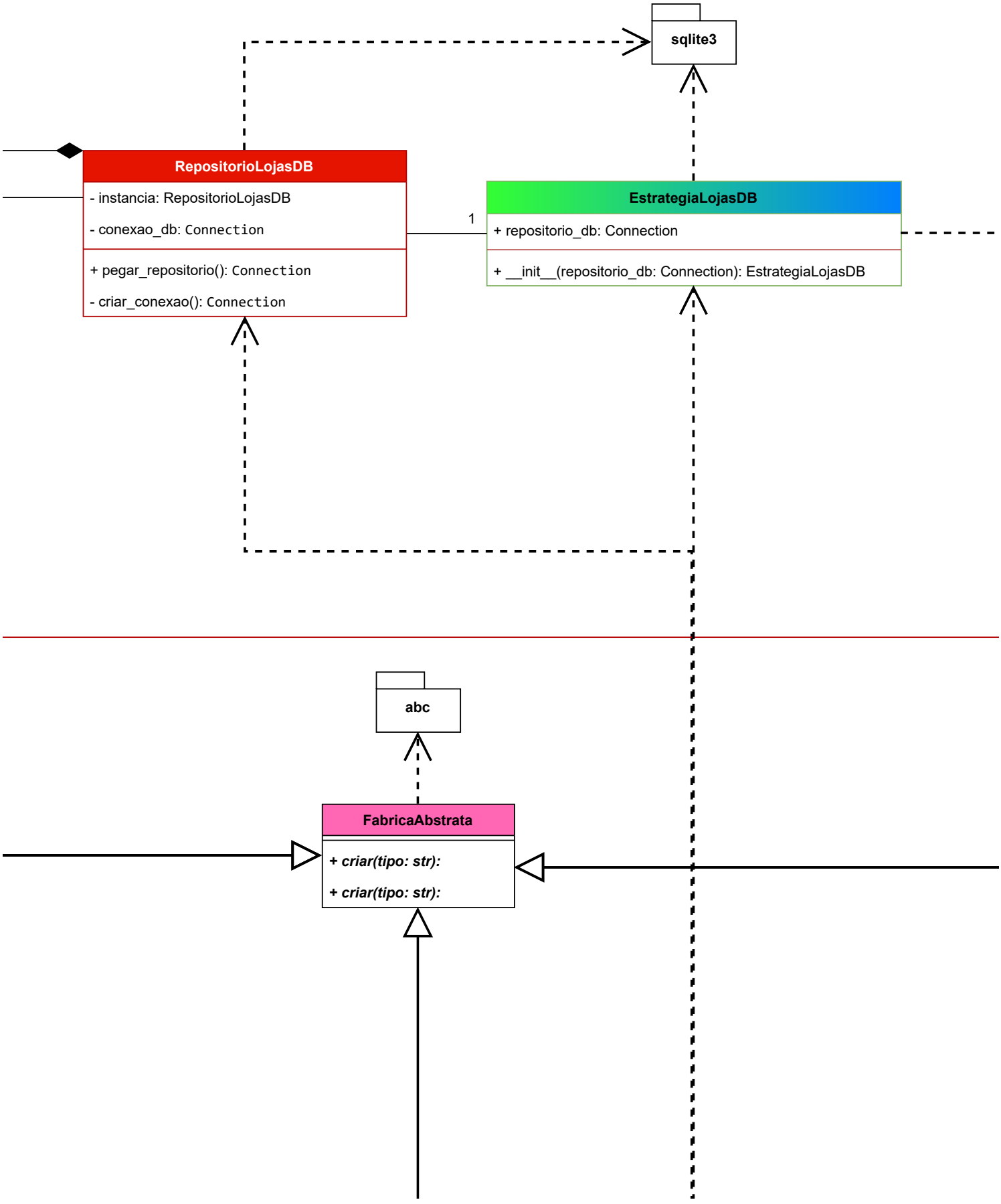

RepositorioNotificacoesFirebase
- instancia: RepositorioNotificacoesFirebase
- repositorio: list[Notificacao]
+ pegar_repositorio(): list[Notificacao]

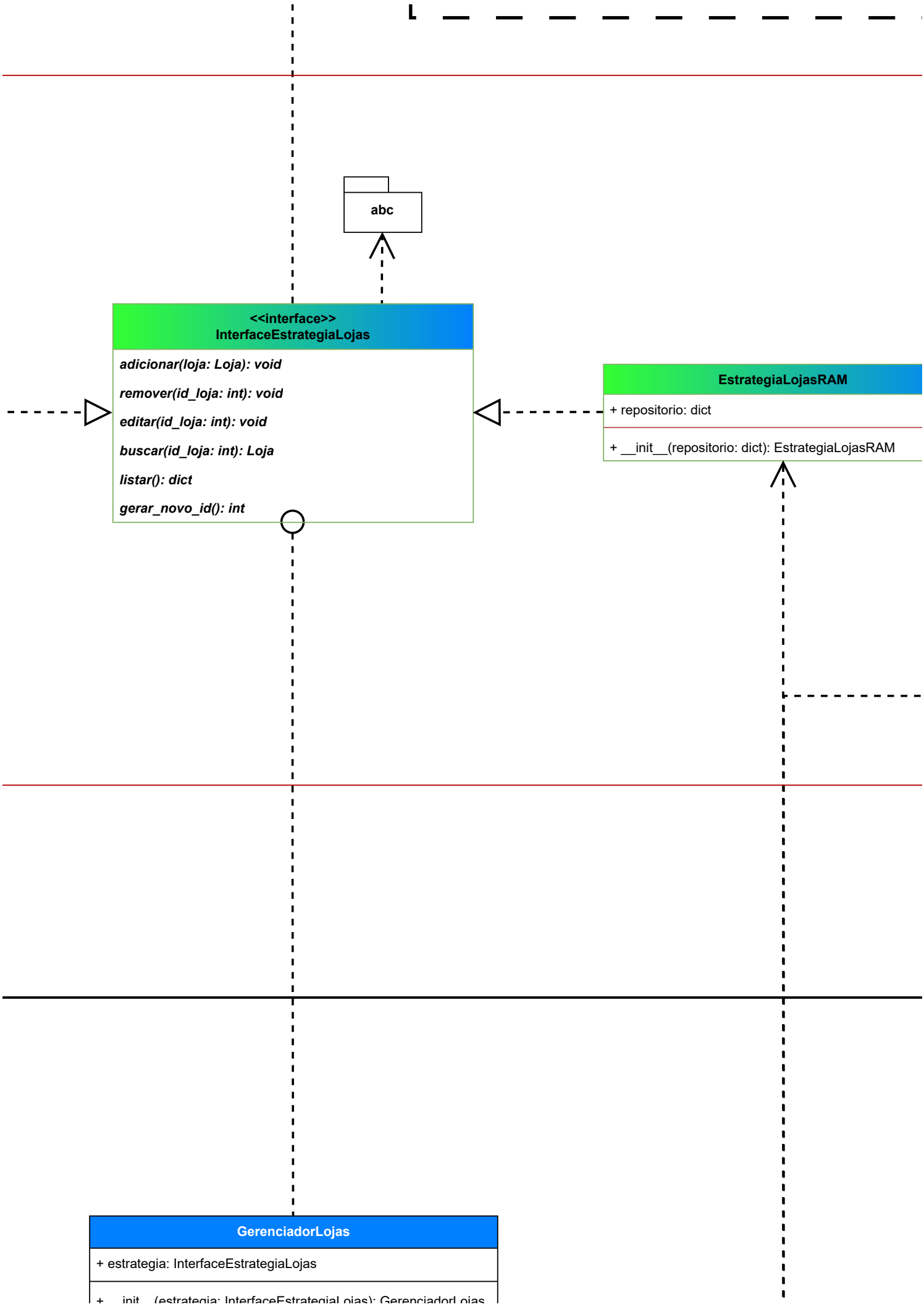

sqlite3
RepositorioUsuariosDB
- instancia: RepositorioUsuariosDB
- conexao_db: Connection
+ pegar_repositorio(): Connection
- criar_conexao(): Connection
1
EstrategiaUsuariosDB
+ repositorio_db: Connection
+ __init__(repositorio_db: Connection): EstrategiaUsuariosDB

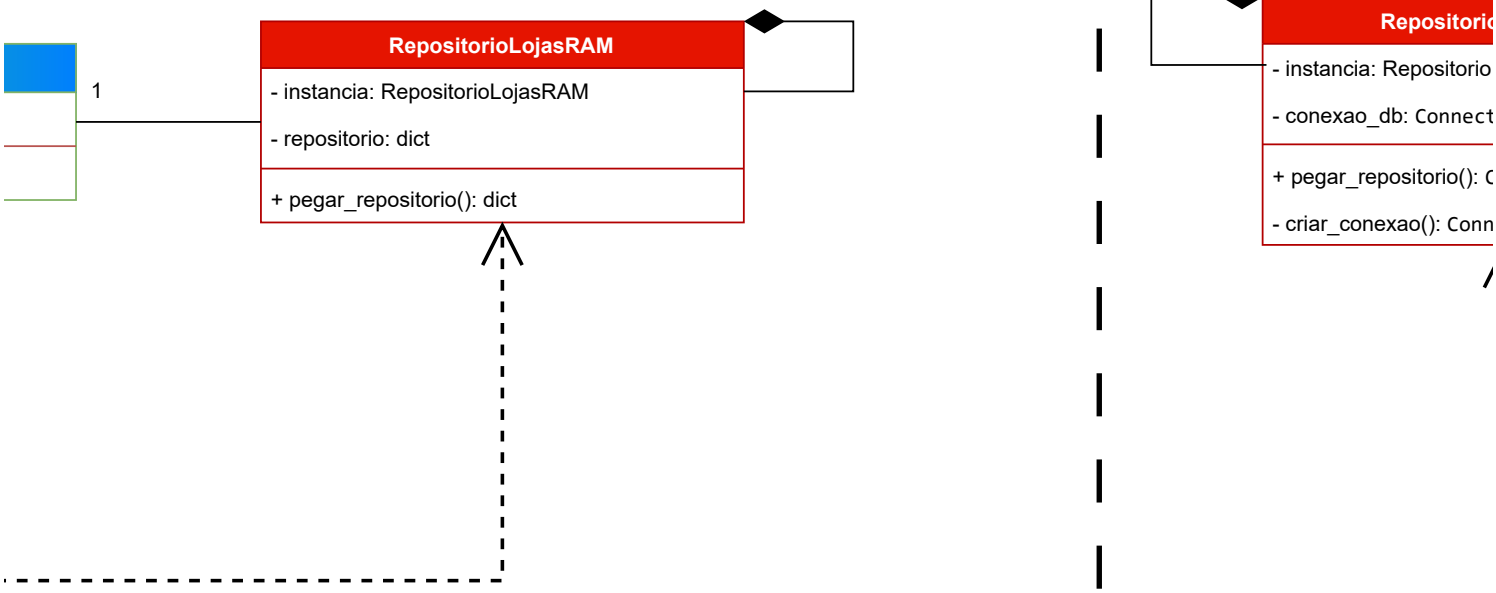

abc
<<interface>>
InterfaceEstrategiaUsuarios
adicionar(usuario: Usuario): void
remover(id_: int): void
editar(id_: int, usuario: Usuario): void
buscar(id_: int): Usuario
listar(tipo: str, id_loja: int): dict
validar(username: str, senha: str): Usuario
gerar_novo_id(): int
EstrategiaUsuariosRAM
+ repositorio: dict
+ __init__(repositorio: dict): EstrategiaUsuariosR
GerenciadorUsuarios
+ estrategia: InterfaceEstrategiaUsuarios
+ init (estrategia:InterfaceEstrategiaUsuarios): GerenciadorUsuarios

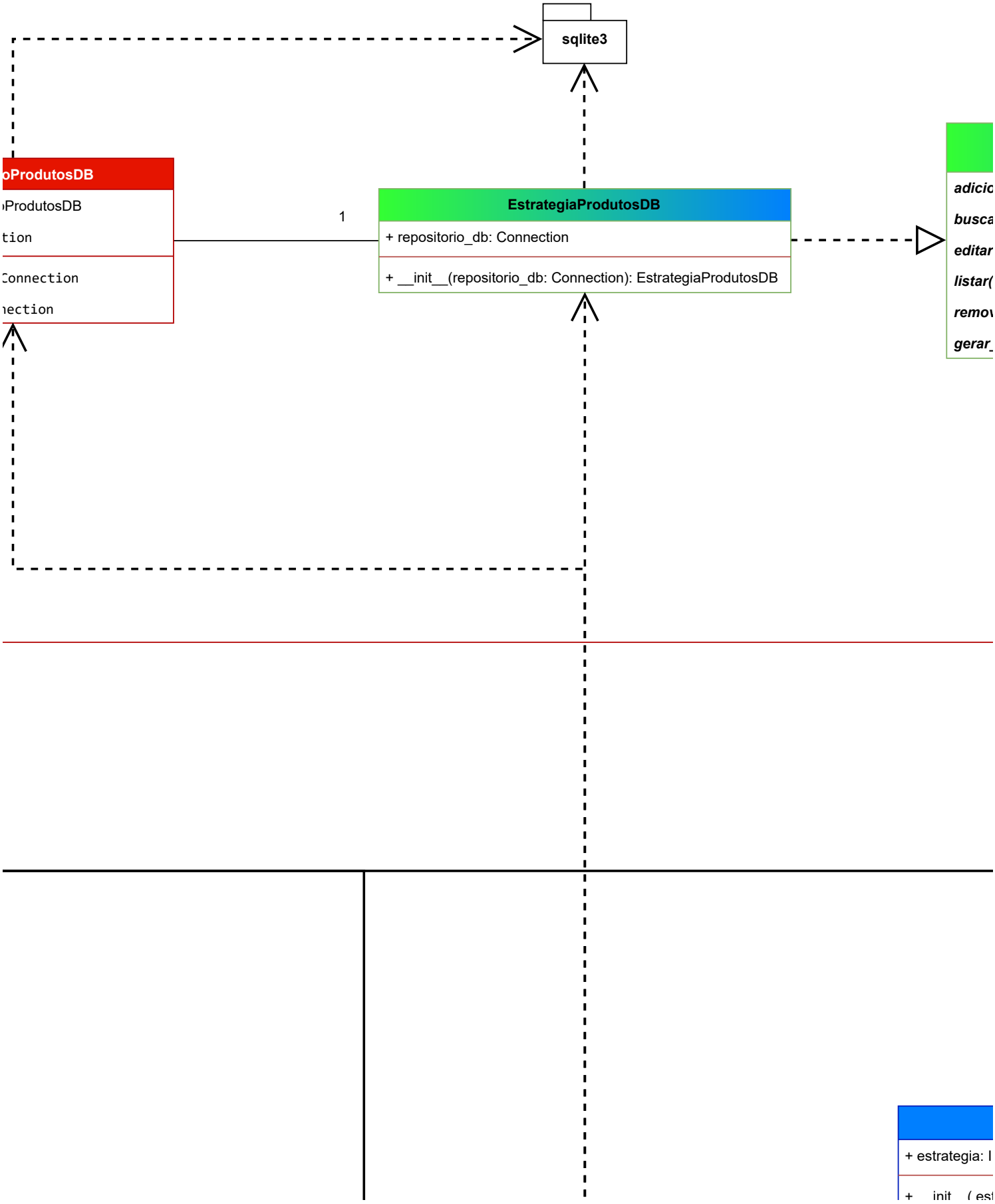

1
AM
RepositorioUsuariosRAM
- instancia: RepositorioUsuariosRAM
- repositorio: dict
+ pegar_repositorio(): dict

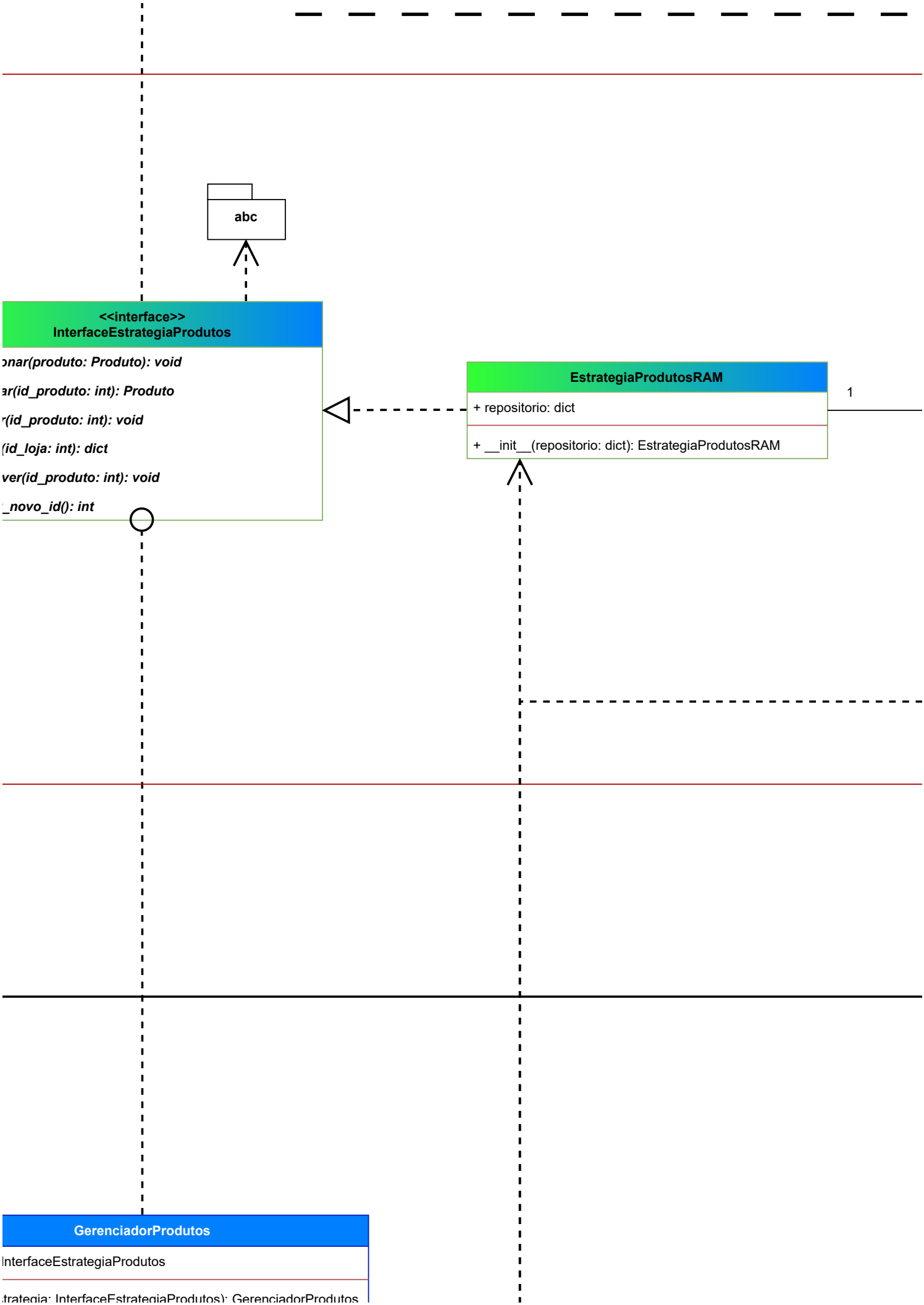

ValidadorInformacoes
+ validacao_username(repositorio: dict, username: str): bool
+ validacao_email(repositorio: dict, email: str): bool
+ validacao_senha(senha: str): bool
+ validacao_nome_loja(repositorio: dict, nome: str): bool
re

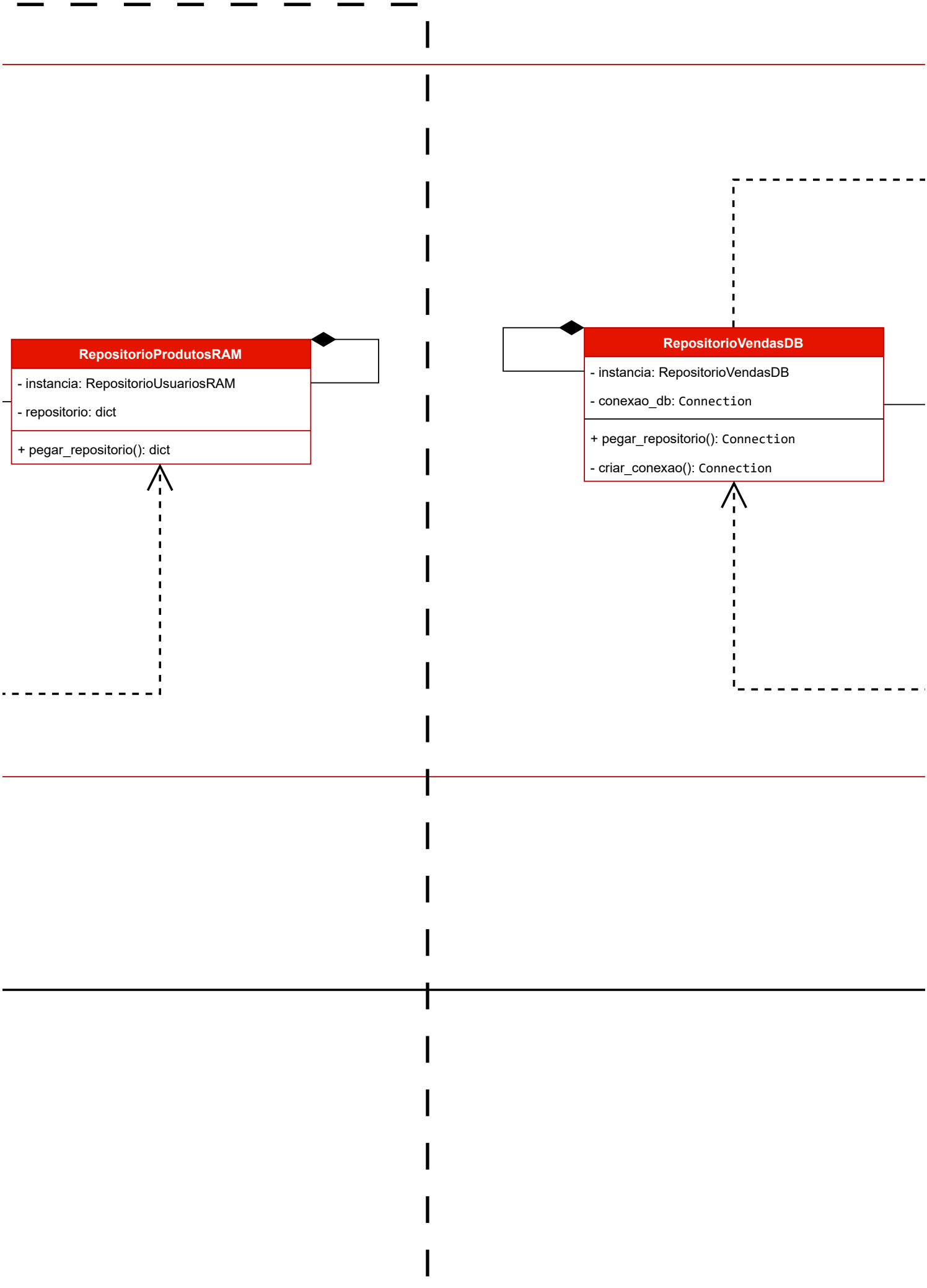

sqlite3
RepositorioLojasDB
- instancia: RepositorioLojasDB
- conexao_db: Connection
+ pegar_repositorio(): Connection
- criar_conexao(): Connection
1
EstrategiaLojasDB
+ repositorio_db: Connection
+ __init__(repositorio_db: Connection): EstrategiaLojasDB
abc
FabricaAbstrata
+ criar(tipo: str):
+ criar(tipo: str):

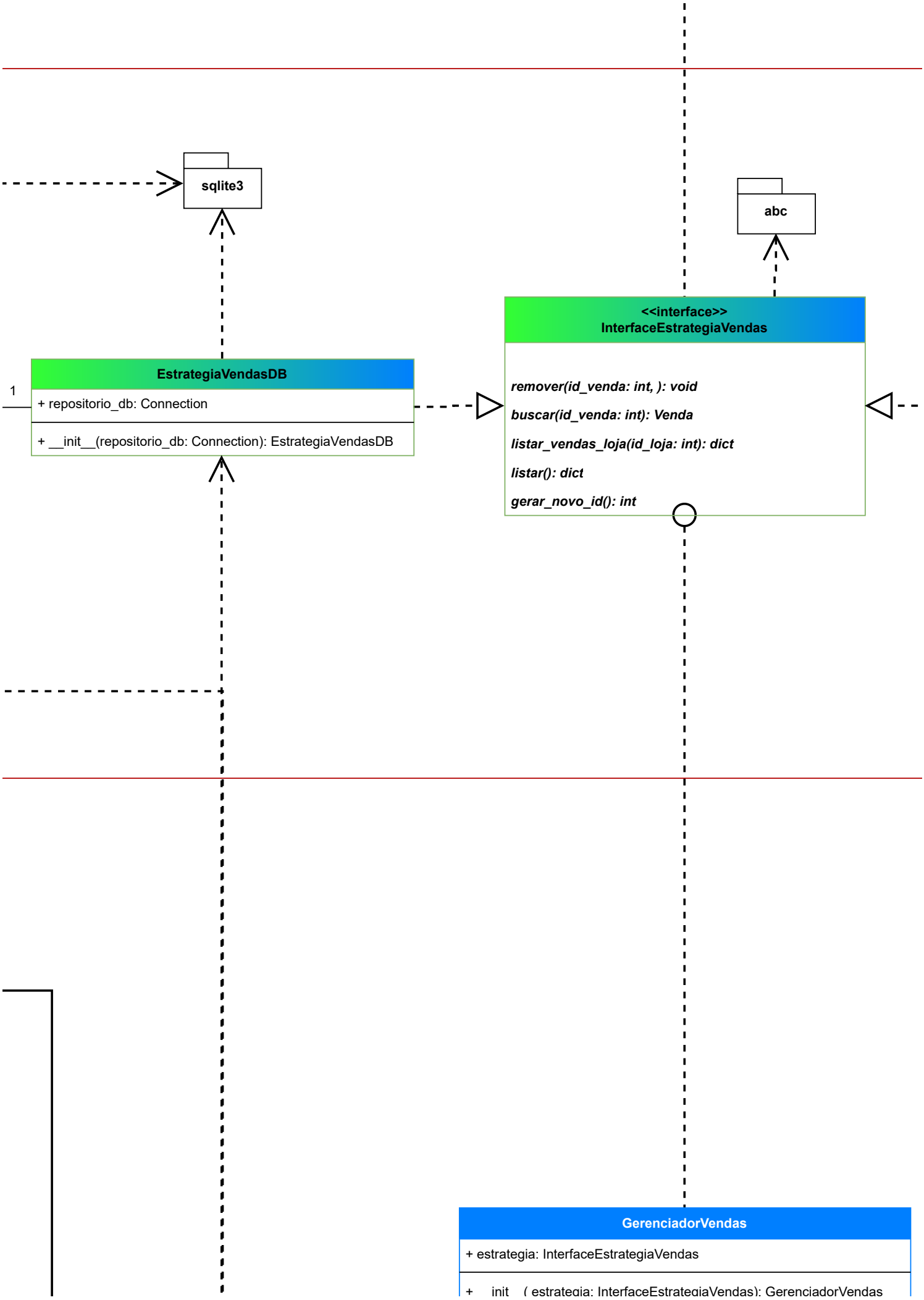

abc
<<interface>>
InterfaceEstrategiaLojas
adicionar(loja: Loja): void
remover(id_loja: int): void
editar(id_loja: int): void
buscar(id_loja: int): Loja
listar(): dict
gerar_novo_id(): int
EstrategiaLojasRAM
+ repositorio: dict
+ __init__(repositorio: dict): EstrategiaLojasRAM
GerenciadorLojas
+ estrategia: InterfaceEstrategiaLojas

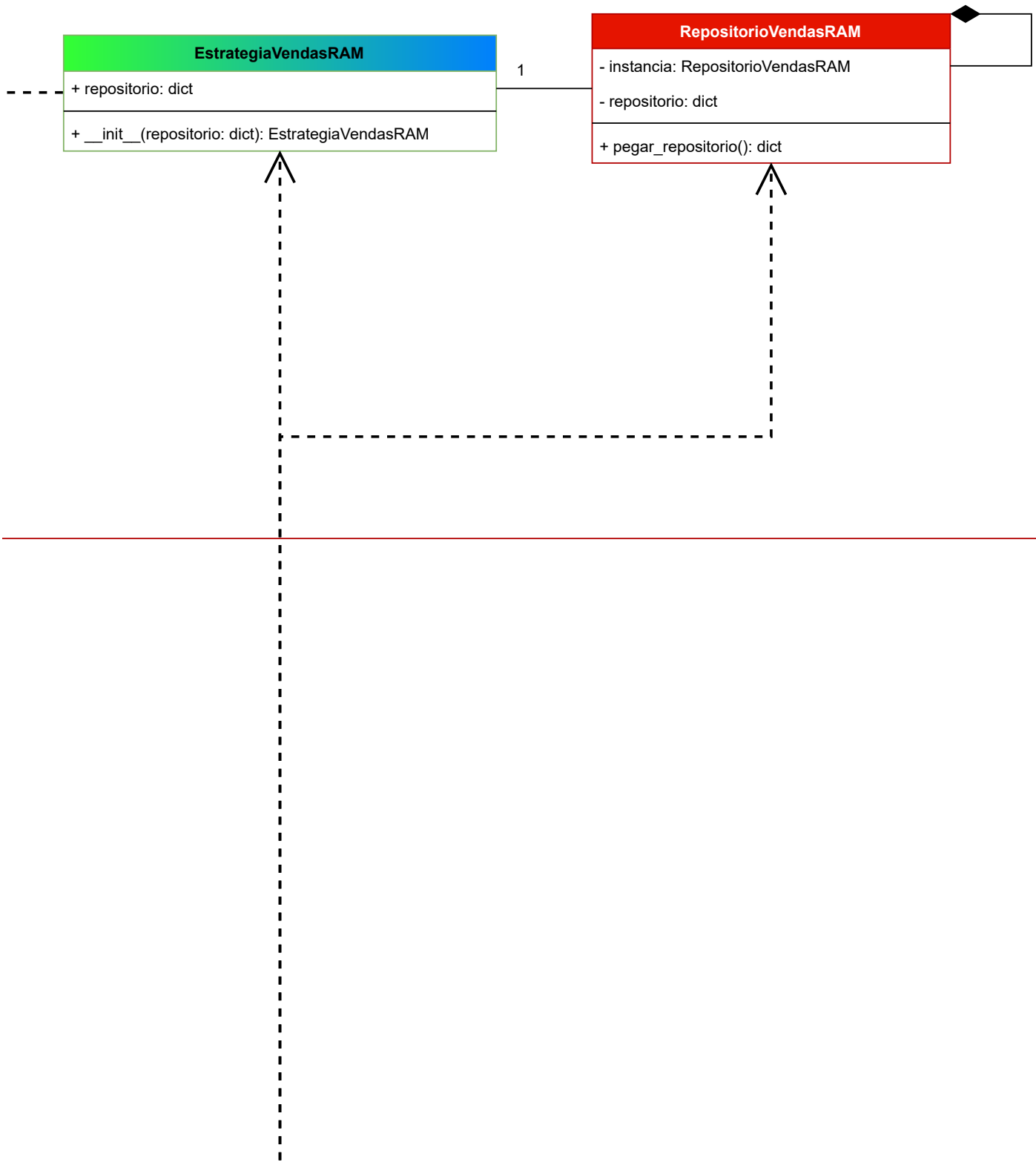

1
RepositorioLojasRAM
- instancia: RepositorioLojasRAM
- repositorio: dict
+ pegar_repositorio(): dict
- instancia: Repositorio
- conexao_db: Connect
+ pegar_repositorio(): C
- criar_conexao(): Conn


sqlite3
oProdutosDB
ProdutosDB
tion
Connection
nection
1
EstrategiaProdutosDB
+ repositorio_db: Connection
+ __init__(repositorio_db: Connection): EstrategiaProdutosDB
adicio
busca
editar
listar(
remov
gerar_
+ estrategia: I
+ init (est


abc
<<interface>>
InterfaceEstrategiaProdutos
onar(produto: Produto): void
ar(id_produto: int): Produto
r(id_produto: int): void
(id_loja: int): dict
ver(id_produto: int): void
_novo_id(): int
EstrategiaProdutosRAM
+ repositorio: dict
+ __init__(repositorio: dict): EstrategiaProdutosRAM
1
GerenciadorProdutos
InterfaceEstrategiaProdutos
trategia: InterfaceEstrategiaProdutos): GerenciadorProdutos


RepositorioProdutosRAM
- instancia: RepositorioUsuariosRAM
- repositorio: dict
+ pegar_repositorio(): dict
RepositorioVendasDB
- instancia: RepositorioVendasDB
- conexao_db: Connection
+ pegar_repositorio(): Connection
- criar_conexao(): Connection

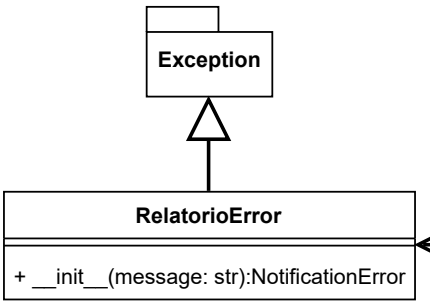

sqlite3
abc
<<interface>>
InterfaceEstrategiaVendas
remover(id_venda: int,): void
buscar(id_venda: int): Venda
listar_vendas_loja(id_loja: int): dict
listar(): dict
gerar_novo_id(): int
1
EstrategiaVendasDB
+ repositorio_db: Connection
+ __init__(repositorio_db: Connection): EstrategiaVendasDB
GerenciadorVendas
+ estrategia: InterfaceEstrategiaVendas
+ init (estrategia: InterfaceEstrategiaVendas): GerenciadorVendas

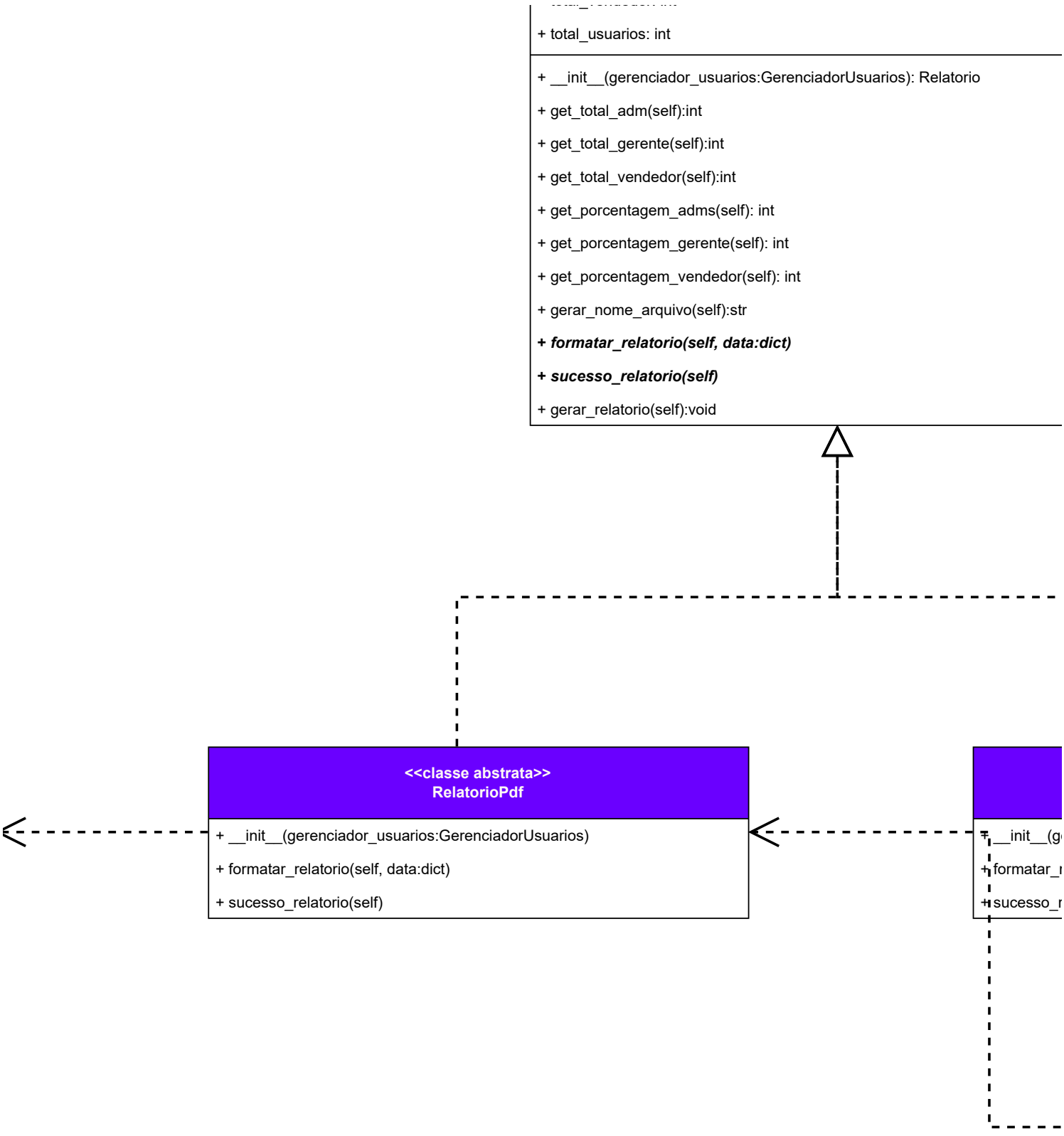

EstrategiaVendasRAM
+ repositorio: dict
+ __init__(repositorio: dict): EstrategiaVendasRAM
1
RepositorioVendasRAM
- instancia: RepositorioVendasRAM
- repositorio: dict
+ pegar_repositorio(): dict

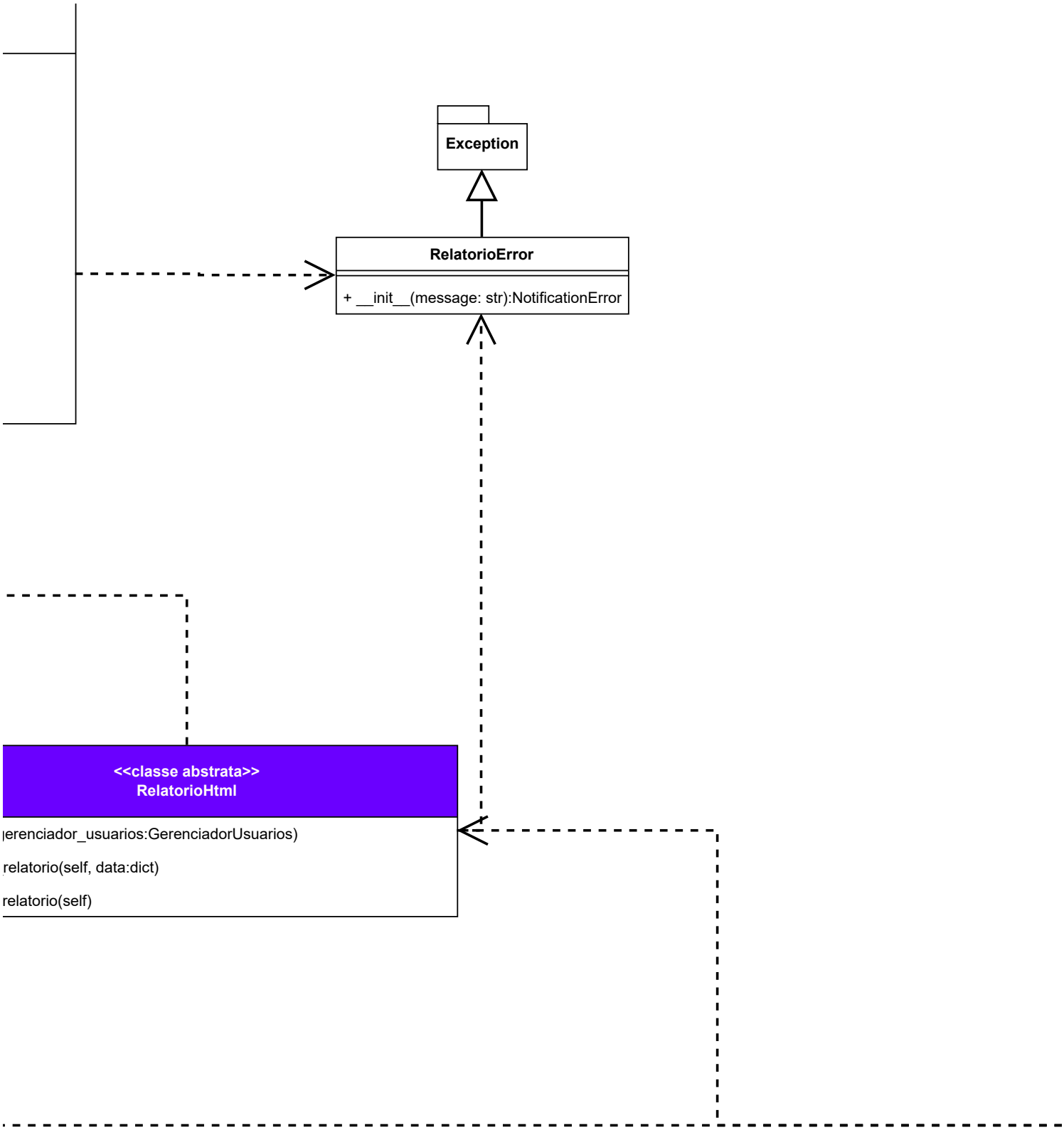
Camada de persistência

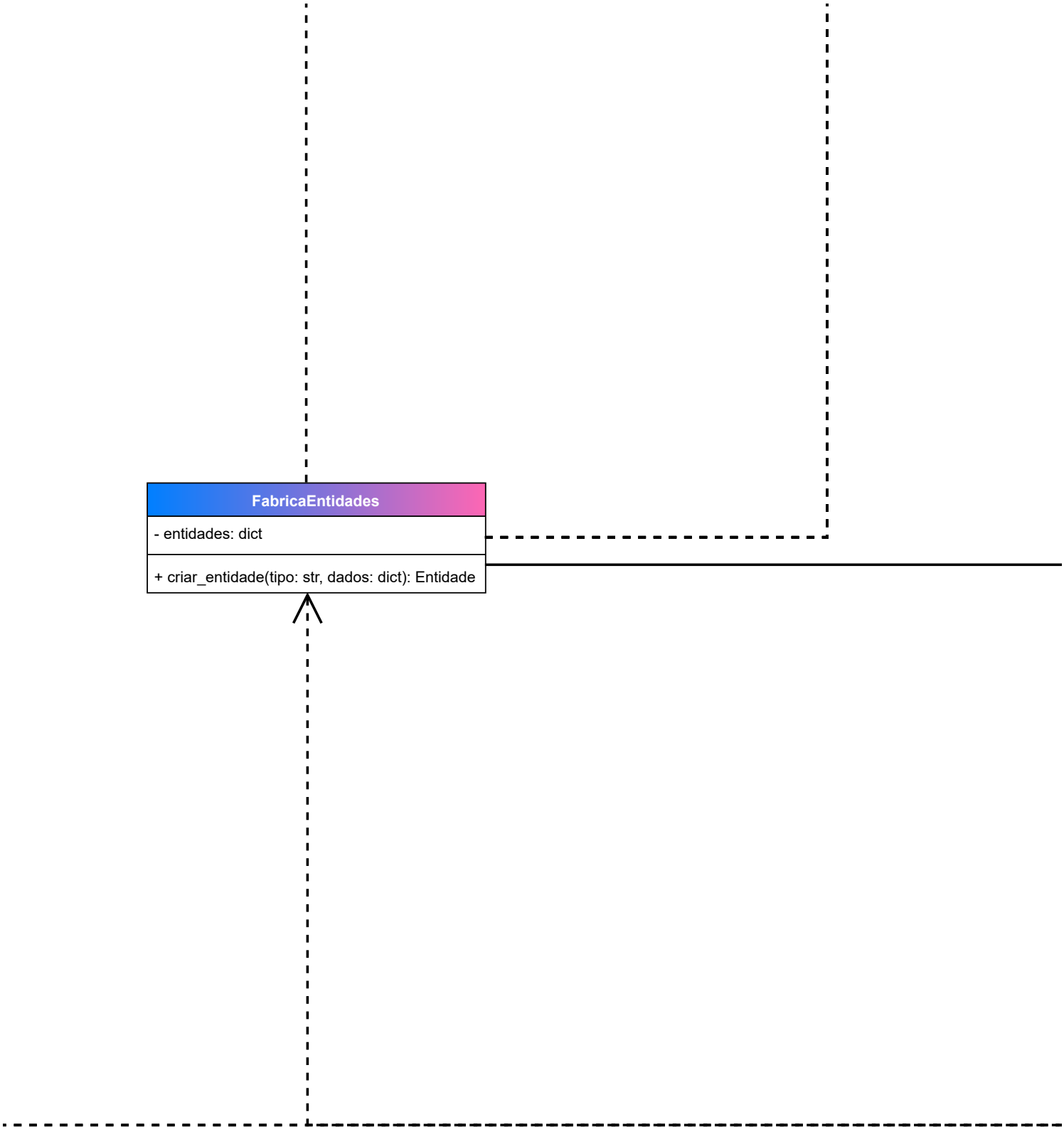
Camada de serviços

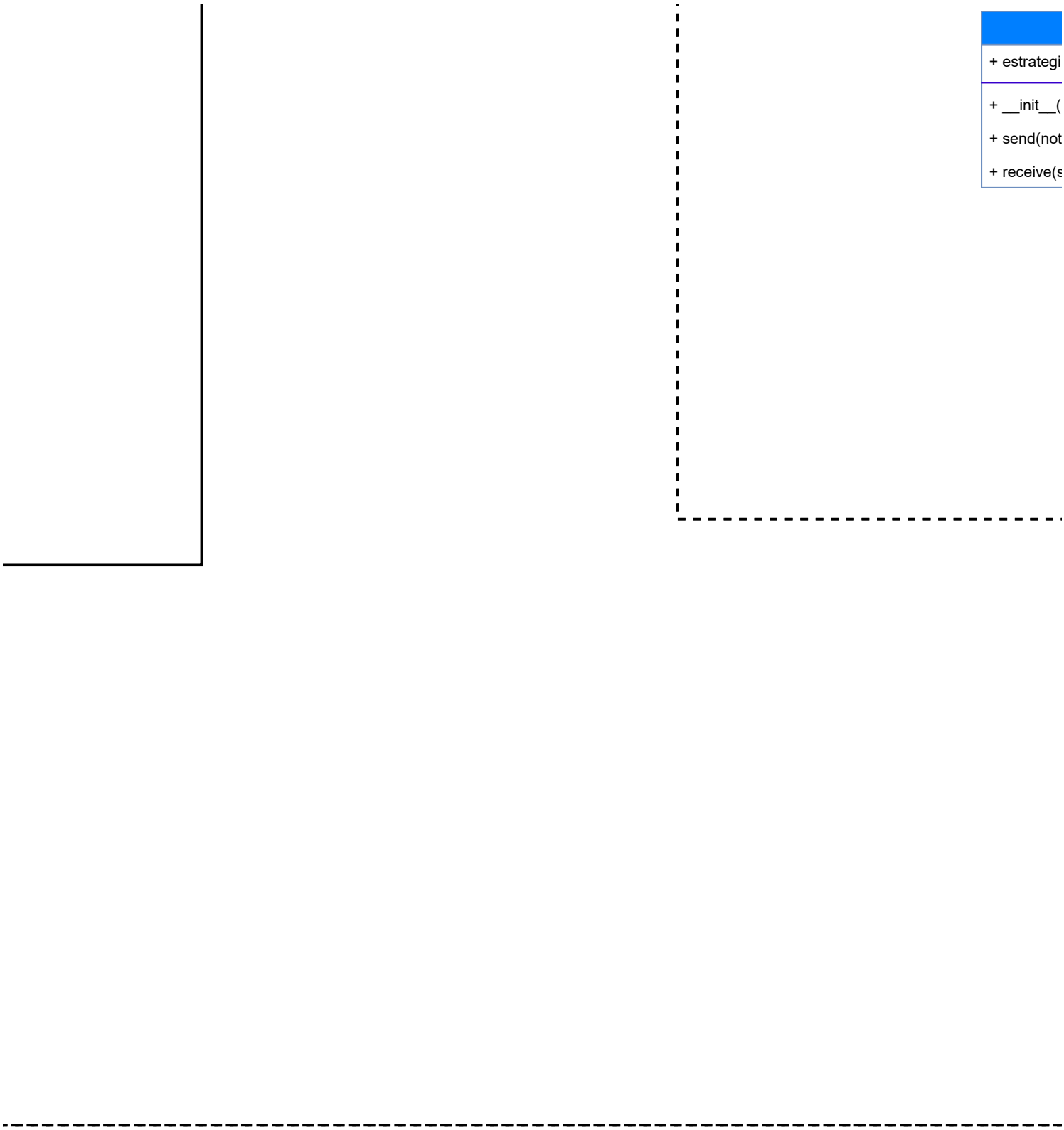
- EnviarNotificacaoComando;
- ExcluirLojaComando;
- ExcluirProdutoComando;
- ExcluirUsuarioComando;
- ComandoLogin;
- VerNotificacoesComando;

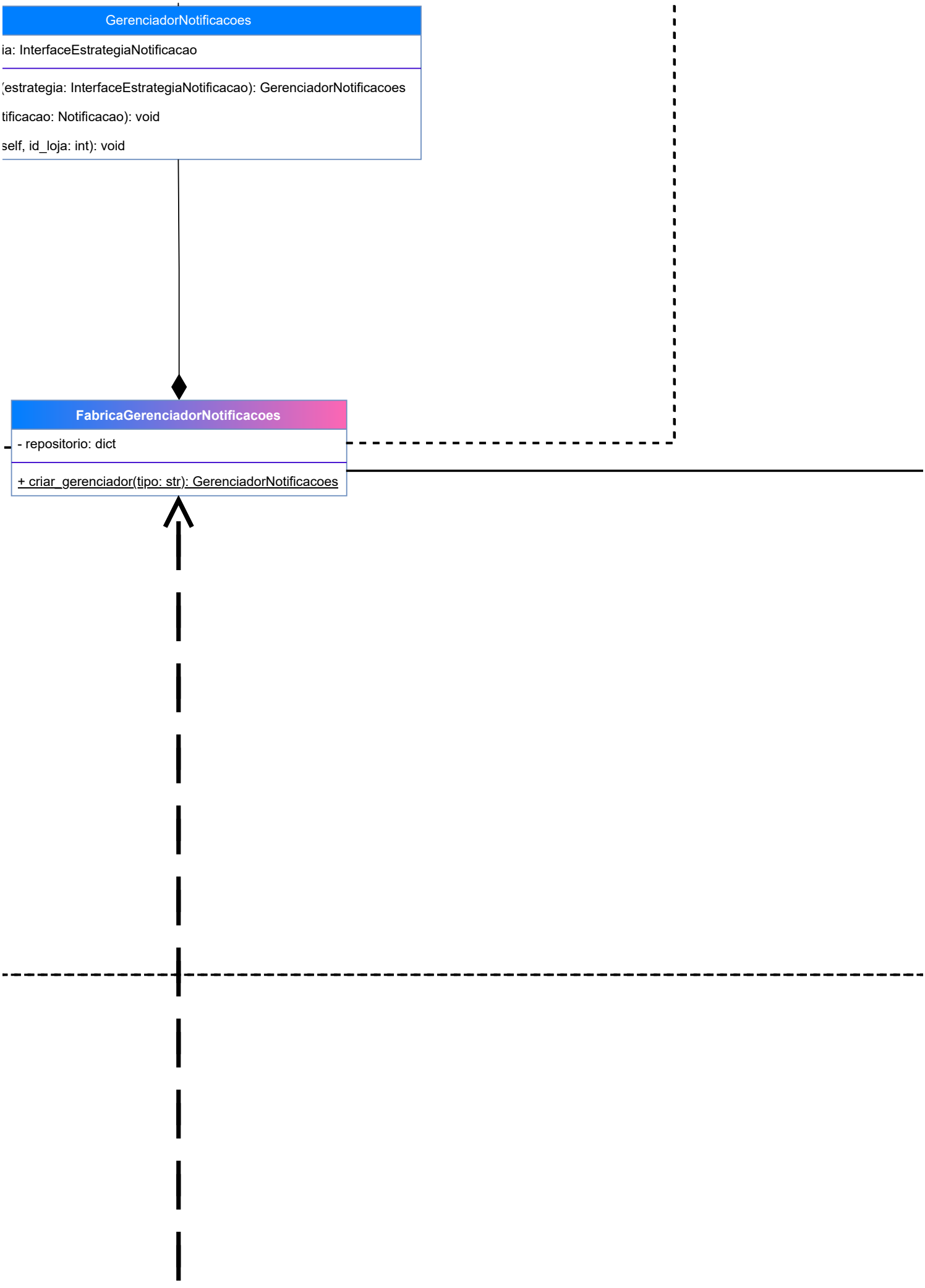

Exception
RelatorioError
+ __init__(message: str):NotificationError

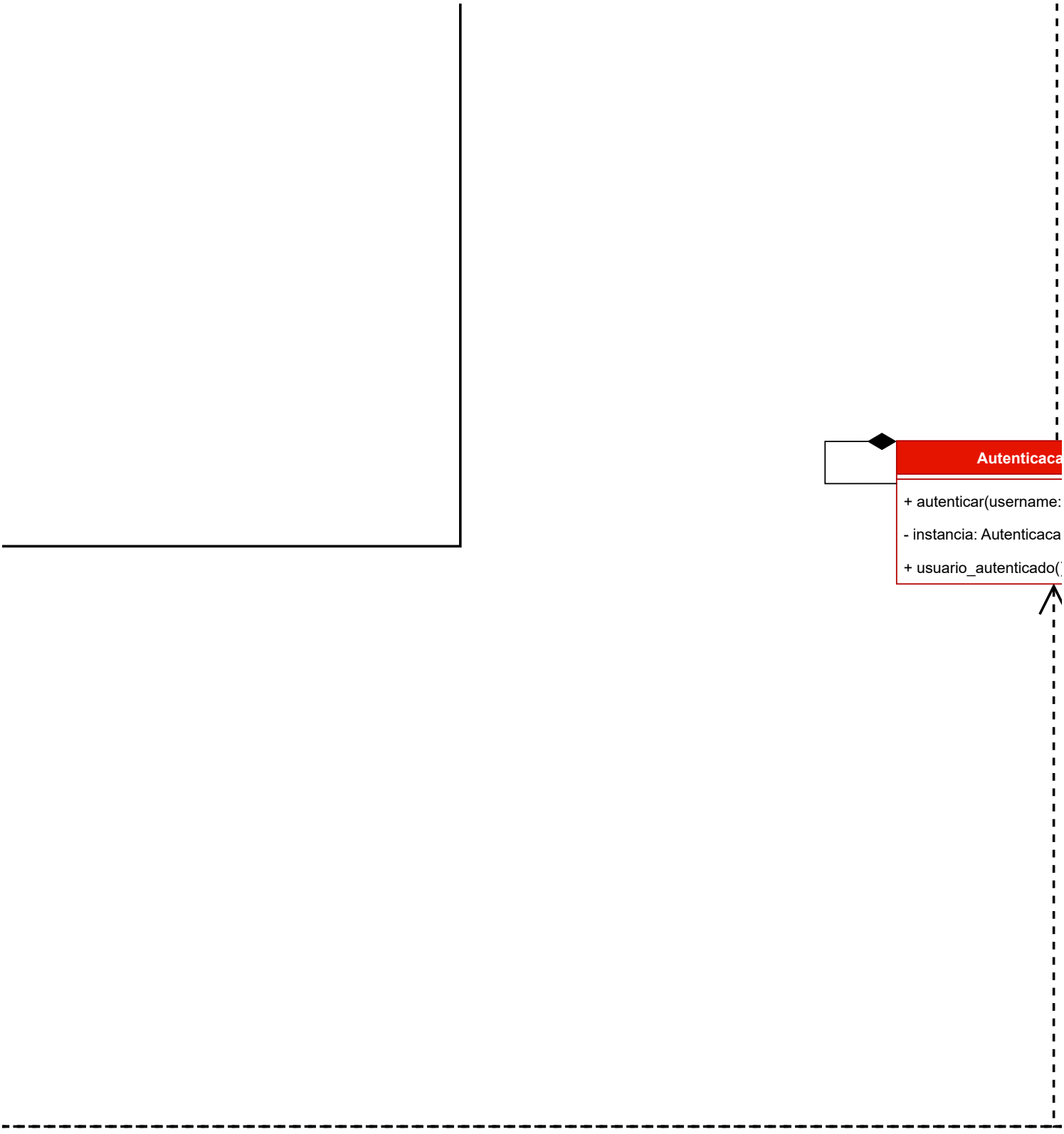

+ total_usuarios: int
+ __init__(gerenciador_usuarios:GerenciadorUsuarios): Relatorio
+ get_total_adm(self):int
+ get_total_gerente(self):int
+ get_total_vendedor(self):int
+ get_porcentagem_adms(self): int
+ get_porcentagem_gerente(self): int
+ get_porcentagem_vendedor(self): int
+ gerar_nome_arquivo(self):str
+ formatar_relatorio(self, data:dict)
+ sucesso_relatorio(self)
+ gerar_relatorio(self):void
<<classe abstrata>>
RelatorioPdf
+ __init__(gerenciador_usuarios:GerenciadorUsuarios)
+ formatar_relatorio(self, data:dict)
+ sucesso_relatorio(self)

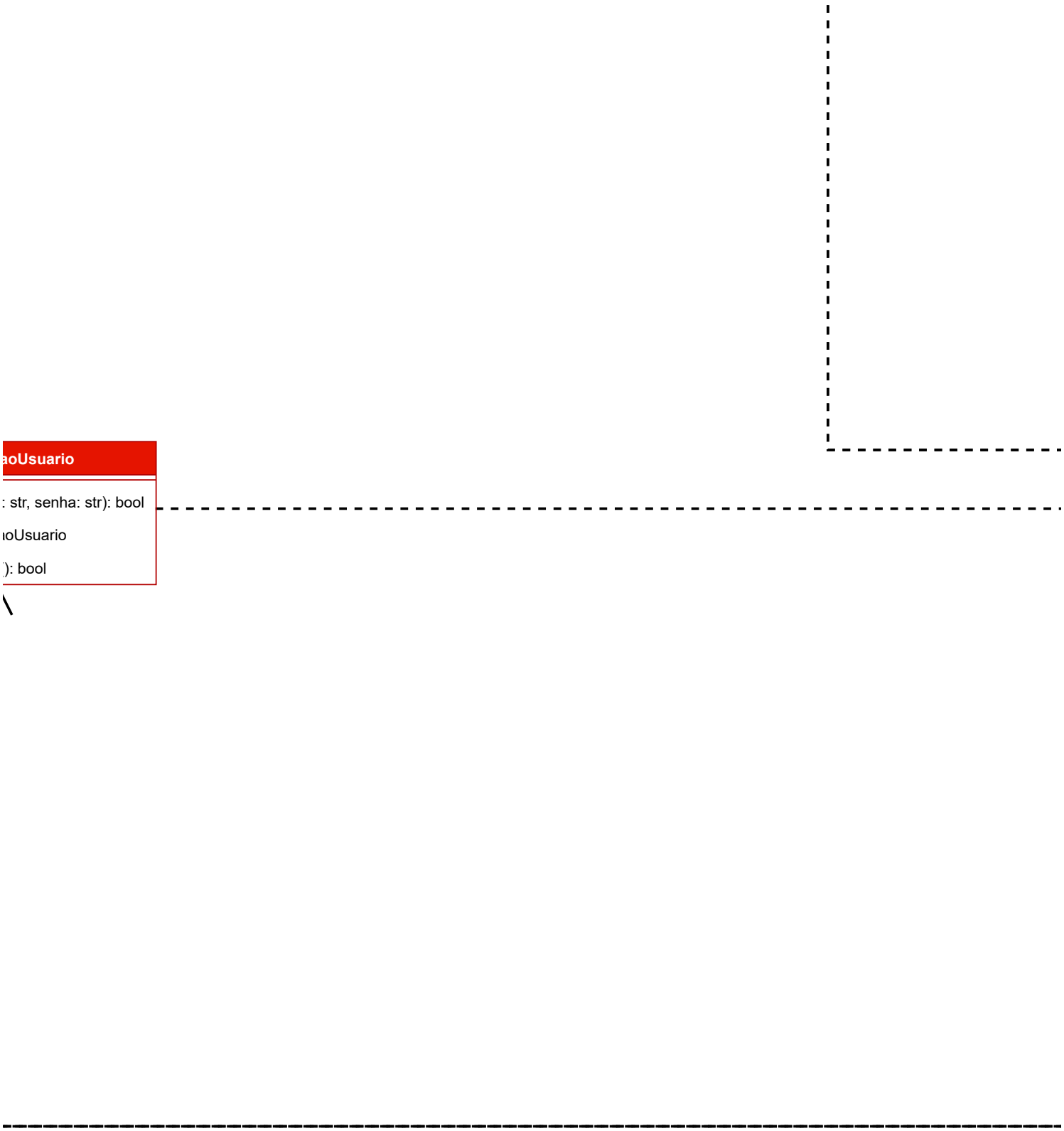

Exception
RelatorioError
+ __init__(message: str):NotificationError
<<classe abstrata>>
RelatorioHtml
erenciador_usuarios:GerenciadorUsuarios)
relatorio(self, data:dict)
relatorio(self)

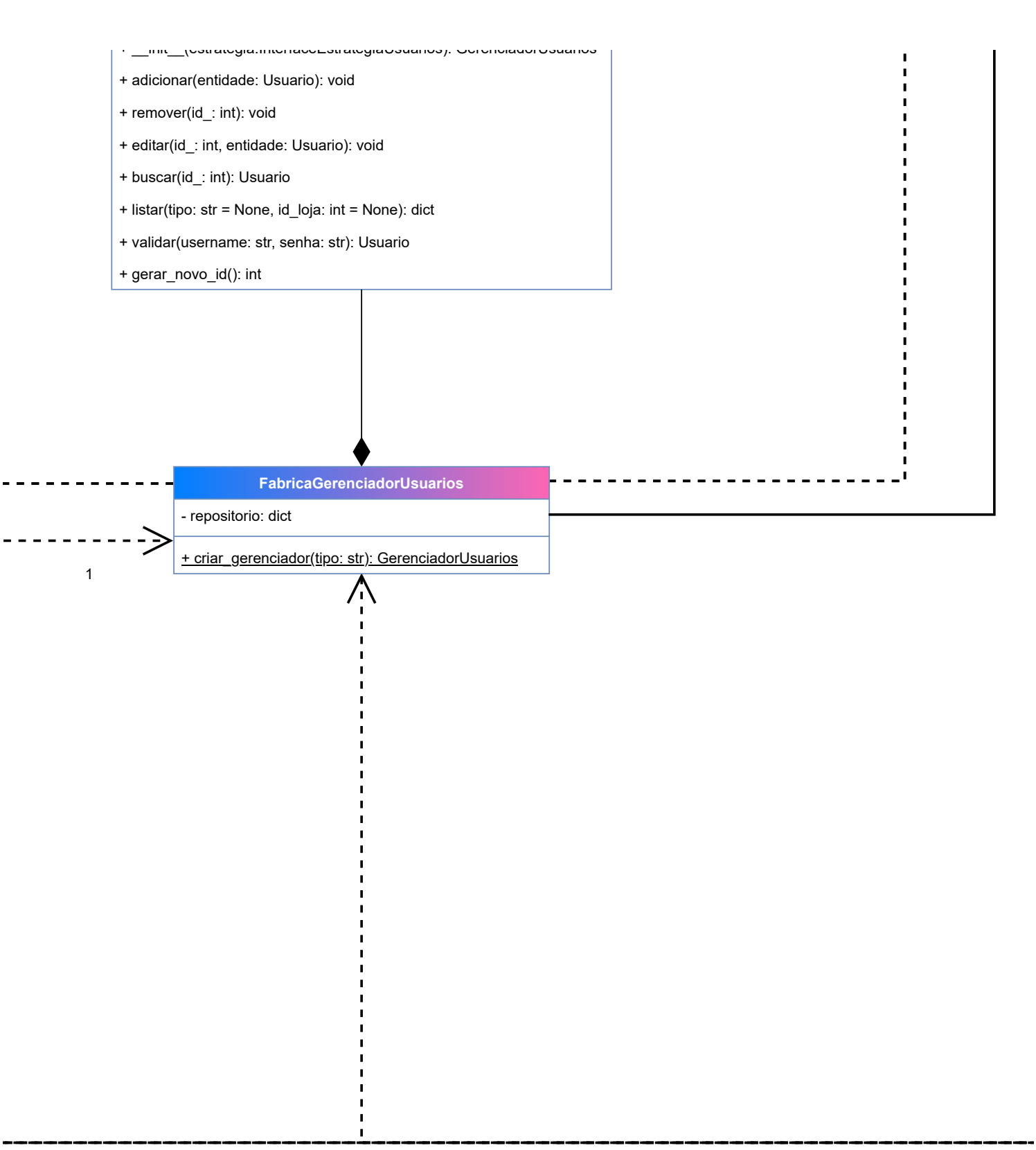

FabricaEntidades
- entidades: dict
+ criar_entidade(tipo: str, dados: dict): Entidade

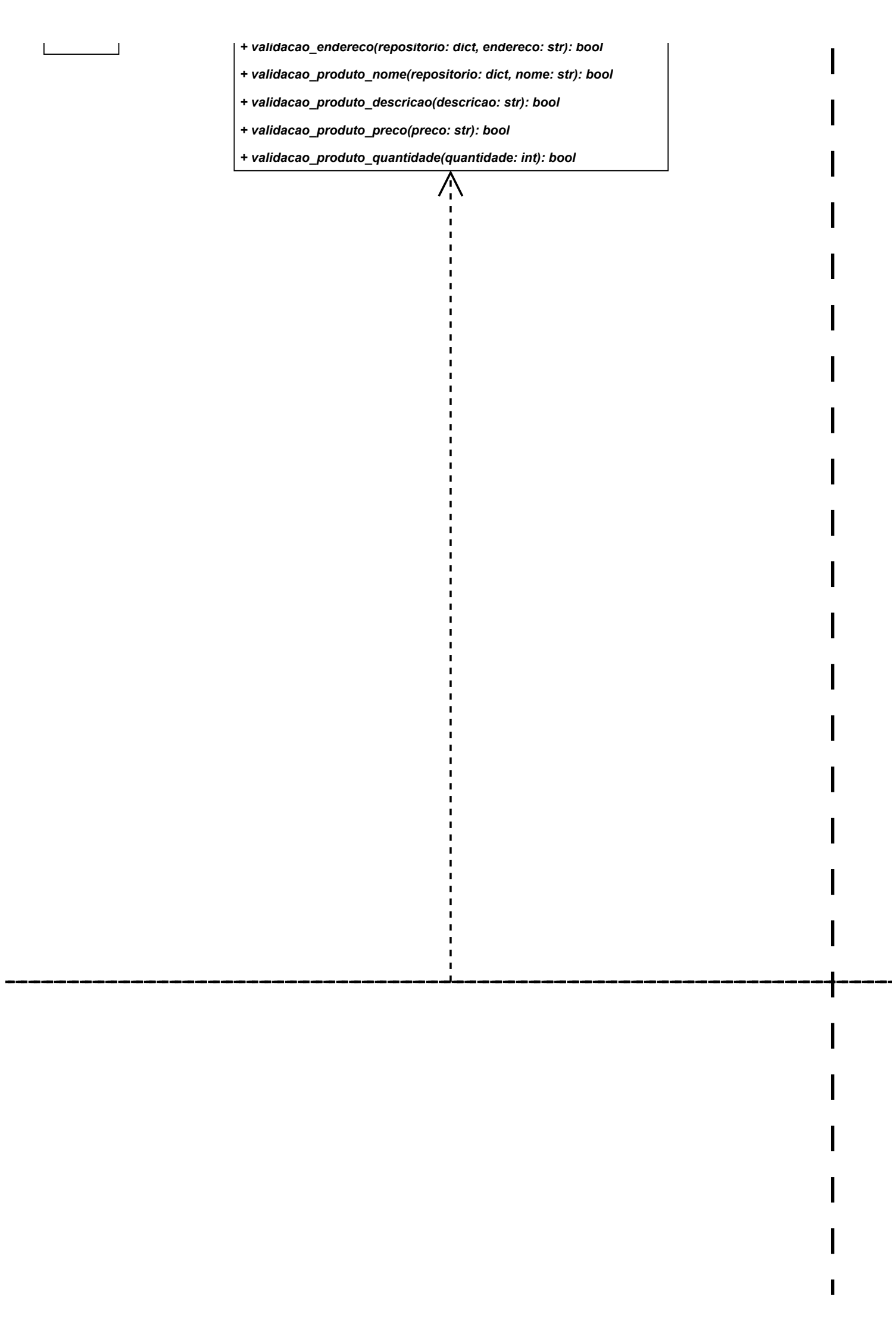

+ estrategi
+ __init__(
+ send(not
+ receive(s

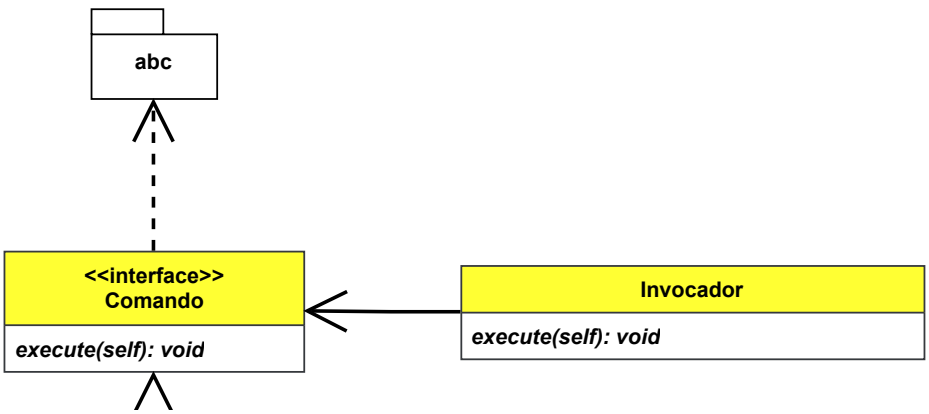

GerenciadorNotificacoes
ia: InterfaceEstrategiaNotificacao
(estrategia: InterfaceEstrategiaNotificacao): GerenciadorNotificacoes
tificacao: Notificacao): void
self, id_loja: int): void
FabricaGerenciadorNotificacoes
- repositorio: dict
+ criar_gerenciador(tipo: str): GerenciadorNotificacoes

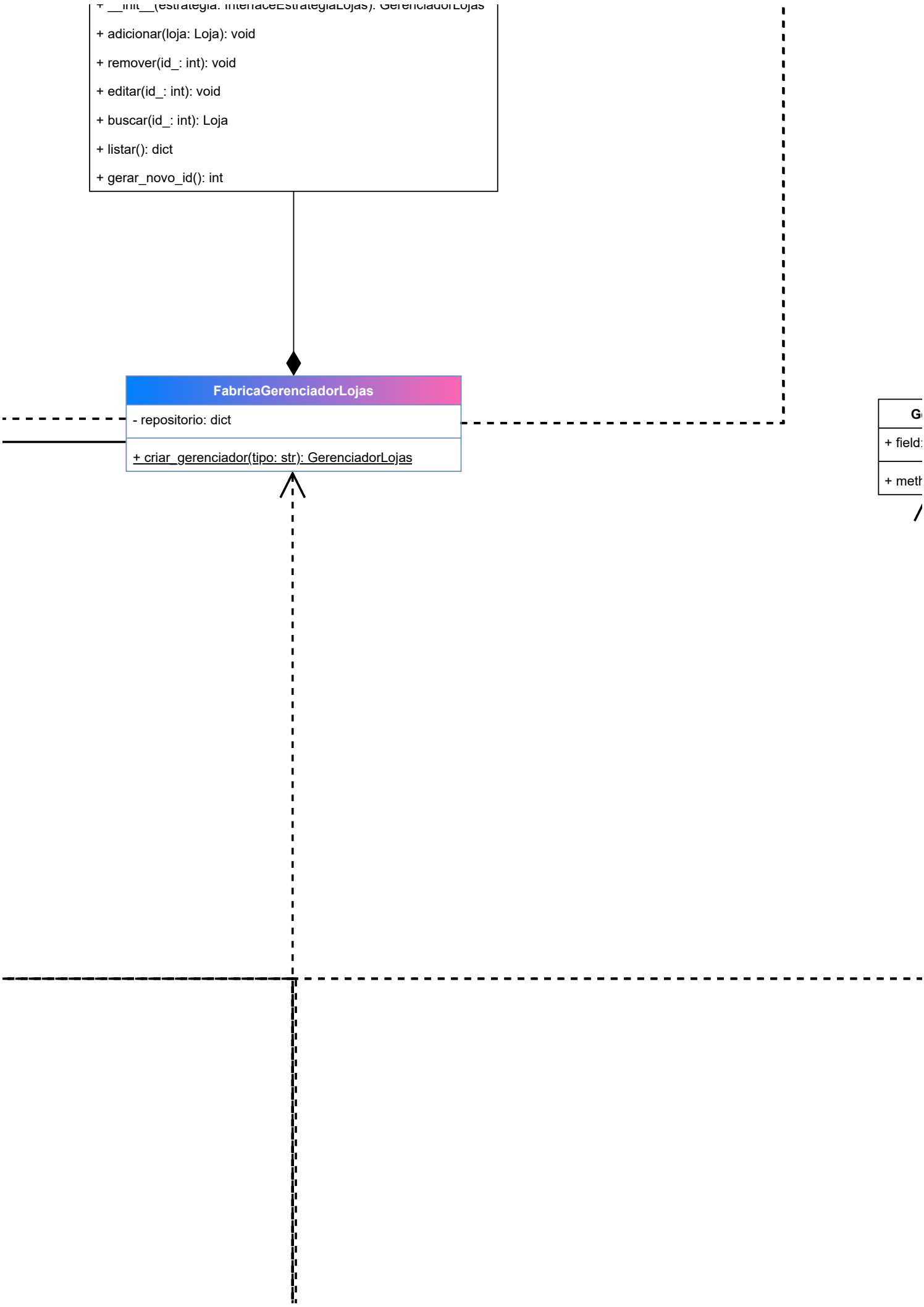

Autenticaca
+ autenticar(username:
- instancia: Autenticaca
+ usuario_autenticado(

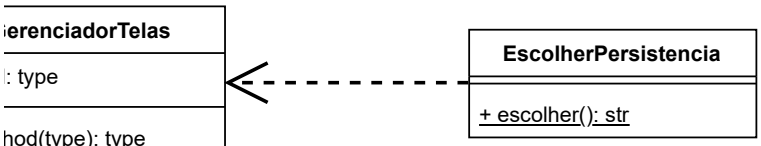

aoUsuario
: str, senha: str): bool
ıoUsuario
): bool

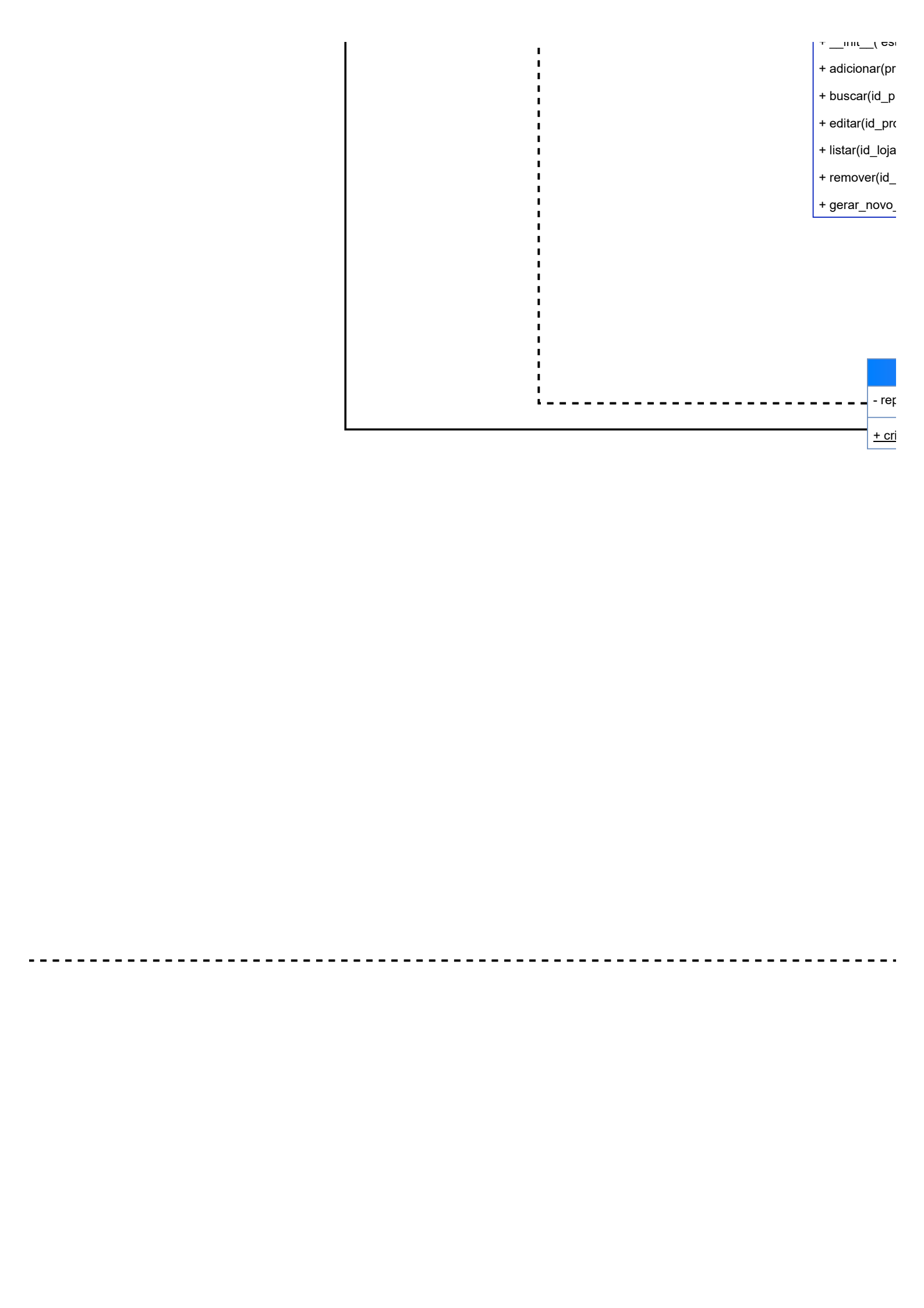

+ adicionar(entidade: Usuario): void
+ remover(id_: int): void
+ editar(id_: int, entidade: Usuario): void
+ buscar(id_: int): Usuario
+ listar(tipo: str = None, id_loja: int = None): dict
+ validar(username: str, senha: str): Usuario
+ gerar_novo_id(): int
FabricaGerenciadorUsuarios
- repositorio: dict
+ criar_gerenciador(tipo: str): GerenciadorUsuarios
1

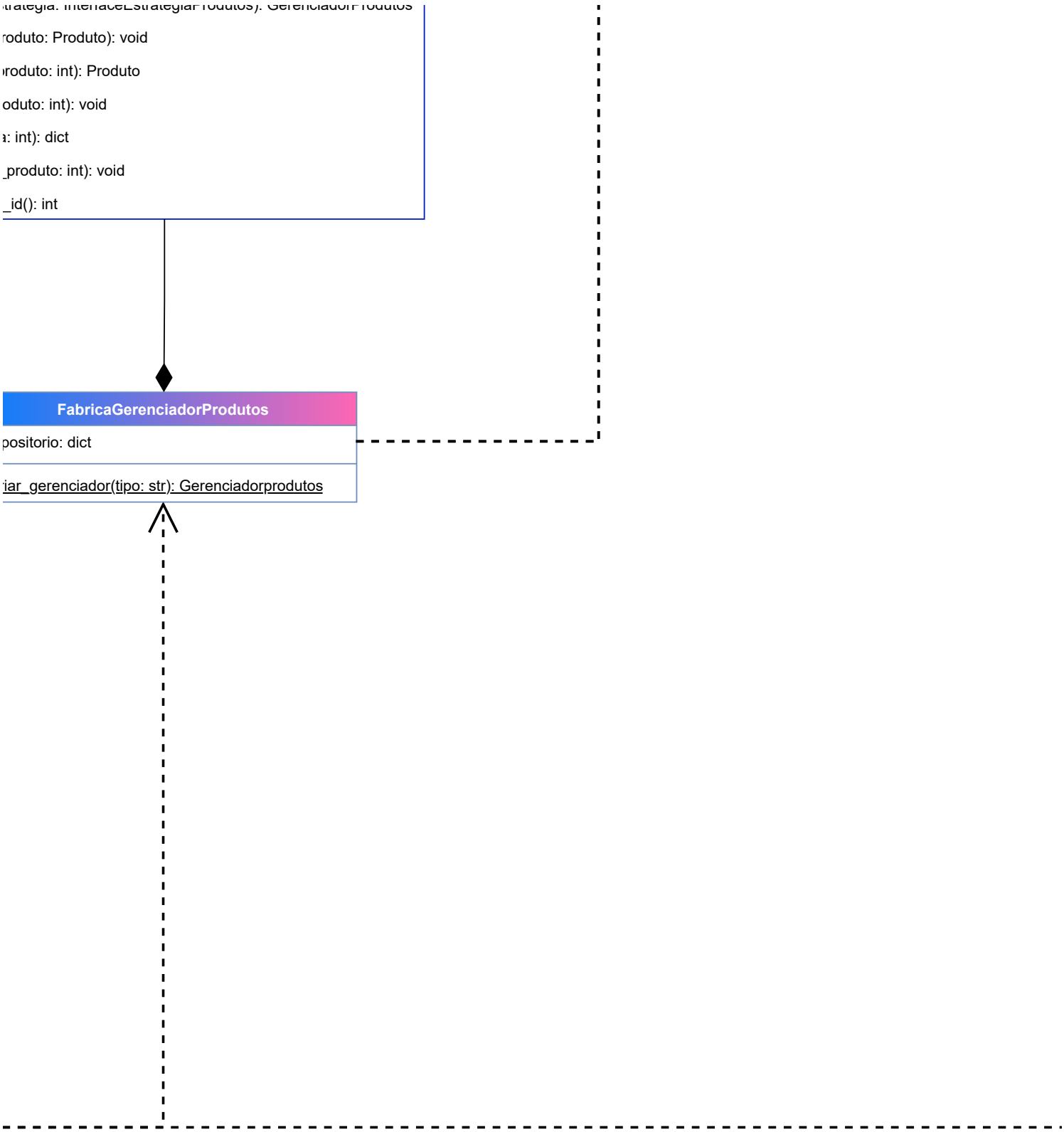

+ validacao_endereco(repositorio: dict, endereco: str): bool
+ validacao_produto_nome(repositorio: dict, nome: str): bool
+ validacao_produto_descricao(descricao: str): bool
+ validacao_produto_preco(preco: str): bool
+ validacao_produto_quantidade(quantidade: int): bool

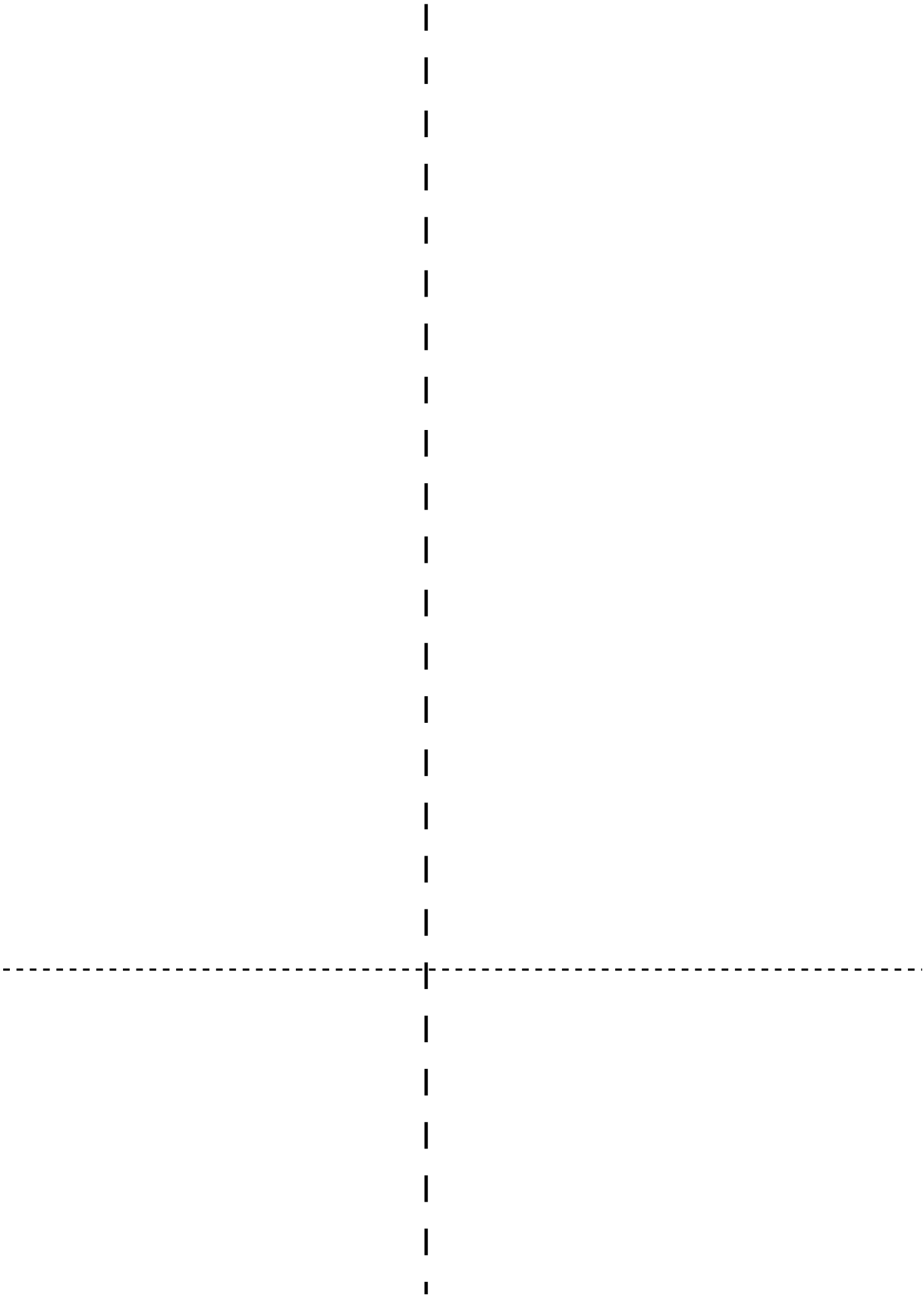

abc
<<interface>>
Comando
execute(self): void
Invocador
execute(self): void

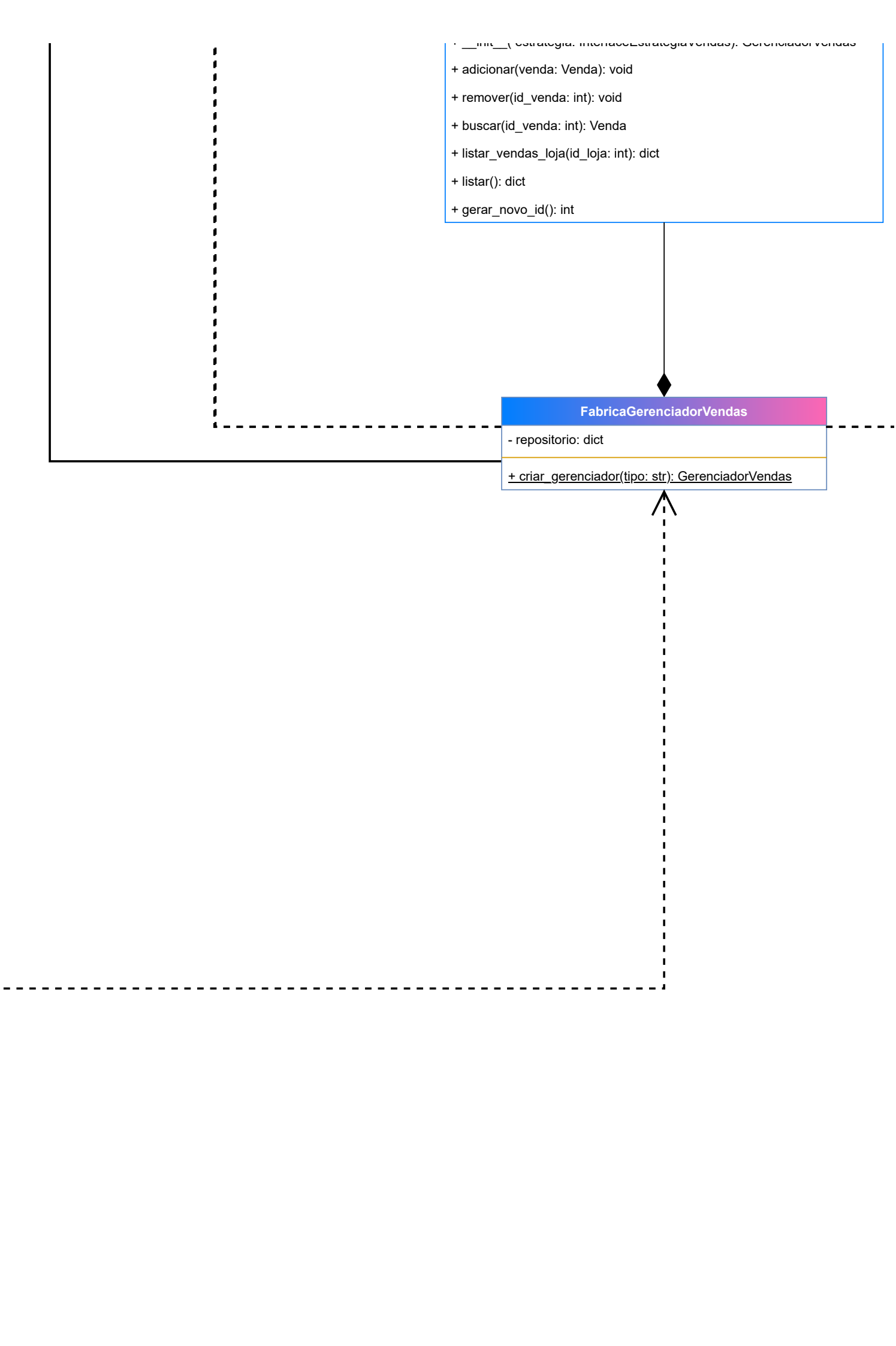

+ adicionar(loja: Loja): void
+ remover(id_: int): void
+ editar(id_: int): void
+ buscar(id_: int): Loja
+ listar(): dict
+ gerar_novo_id(): int
FabricaGerenciadorLojas
- repositorio: dict
+ criar_gerenciador(tipo: str): GerenciadorLojas

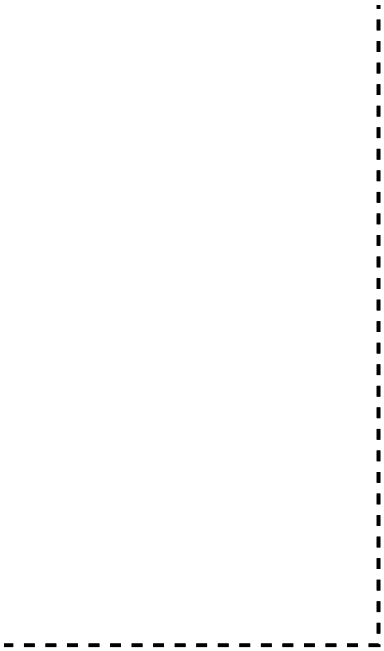

erenciadorTelas
: type
hod(type): type
EscolherPersistencia
+ escolher(): str

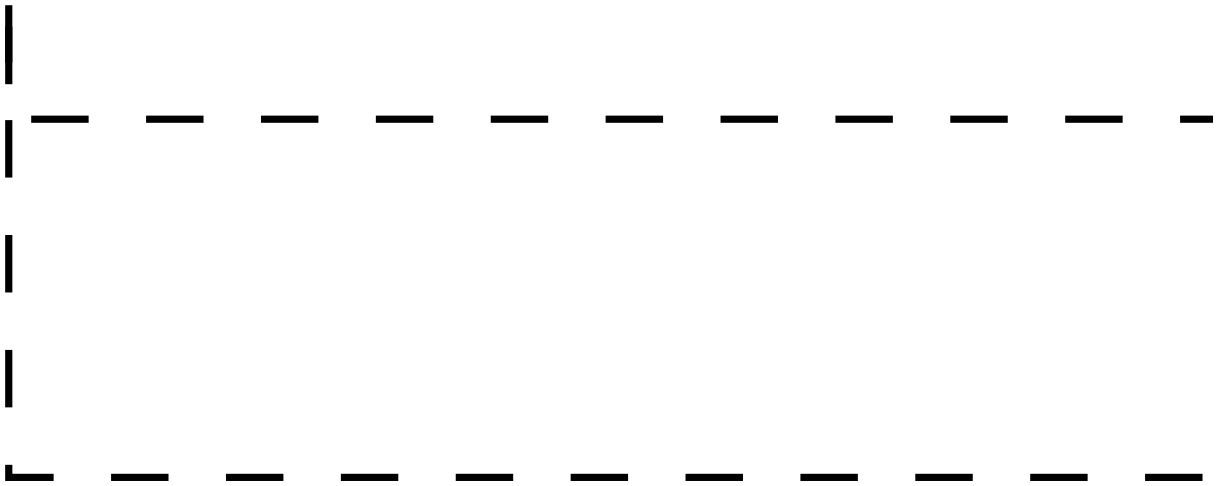

+ adicionar(pr
+ buscar(id_p
+ editar(id_pr
+ listar(id_loja
+ remover(id_
+ gerar_novo_
- rep
+ cri

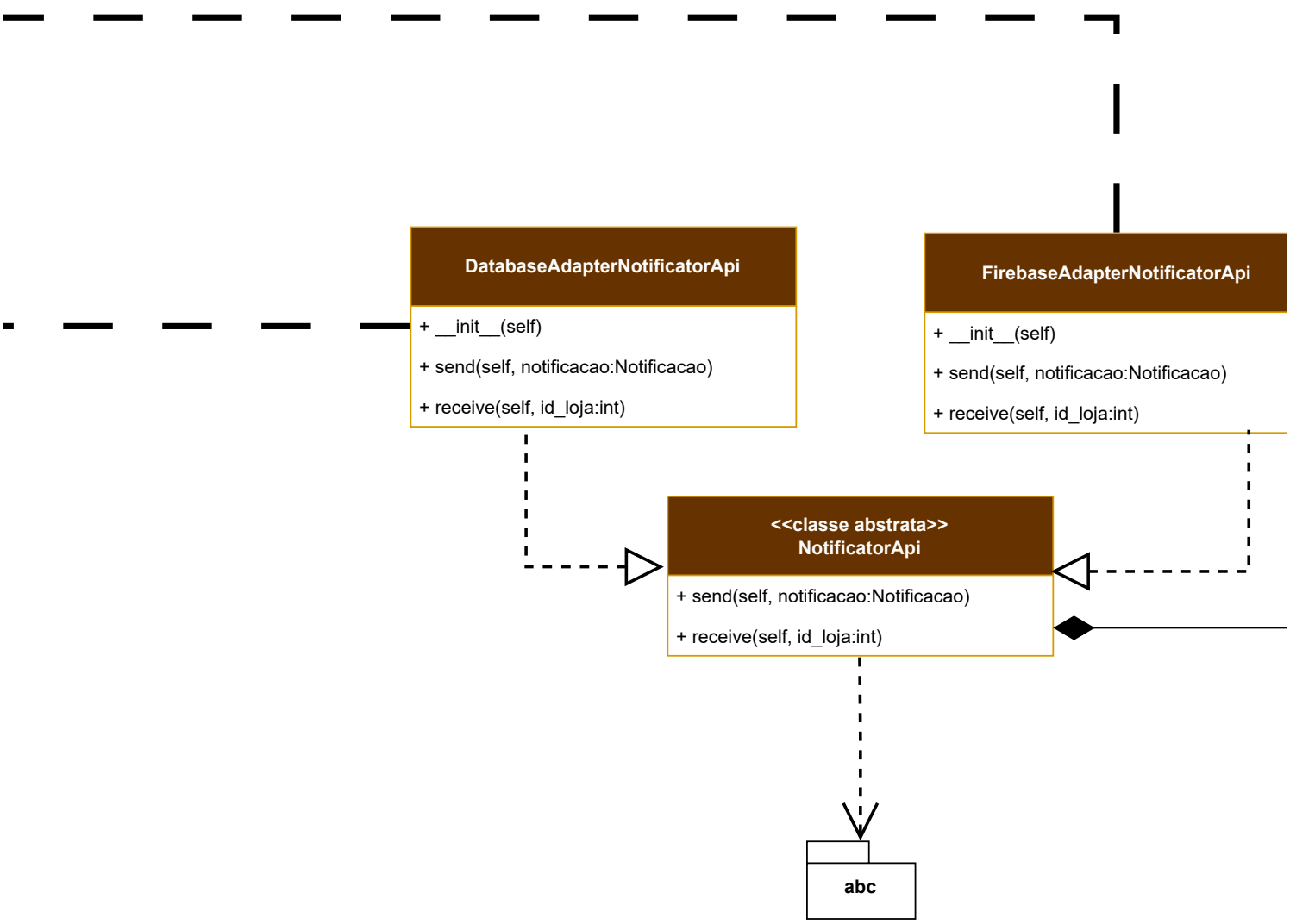

roduto: Produto): void
roduto: int): Produto
oduto: int): void
: int): dict
_produto: int): void
_id(): int
FabricaGerenciadorProdutos
positorio: dict
iar_gerenciador(tipo: str): Gerenciadorprodutos

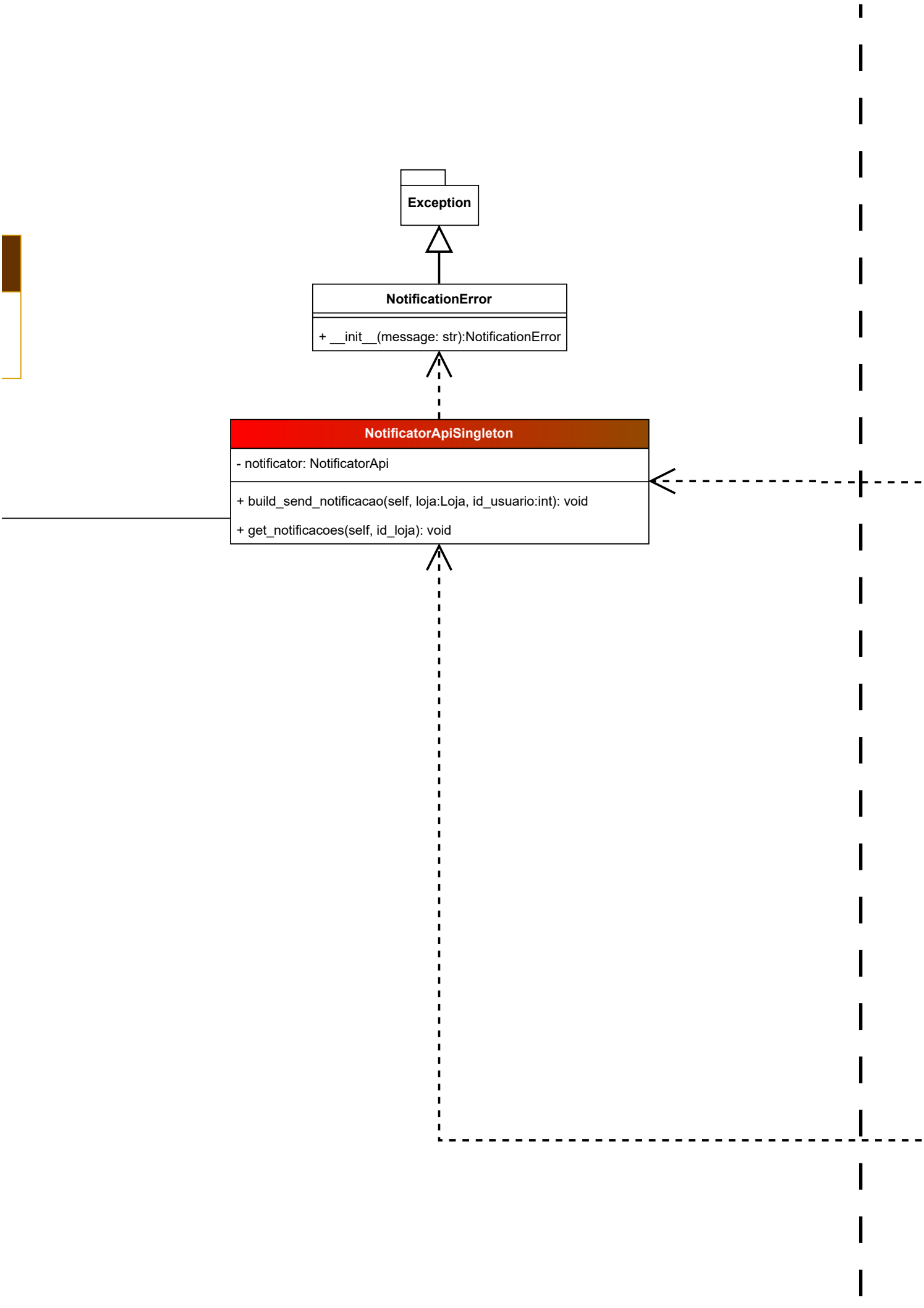



+ adicionar(venda: Venda): void
+ remover(id_venda: int): void
+ buscar(id_venda: int): Venda
+ listar_vendas_loja(id_loja: int): dict
+ listar(): dict
+ gerar_novo_id(): int
FabricaGerenciadorVendas
- repositorio: dict
+ criar_gerenciador(tipo: str): GerenciadorVendas



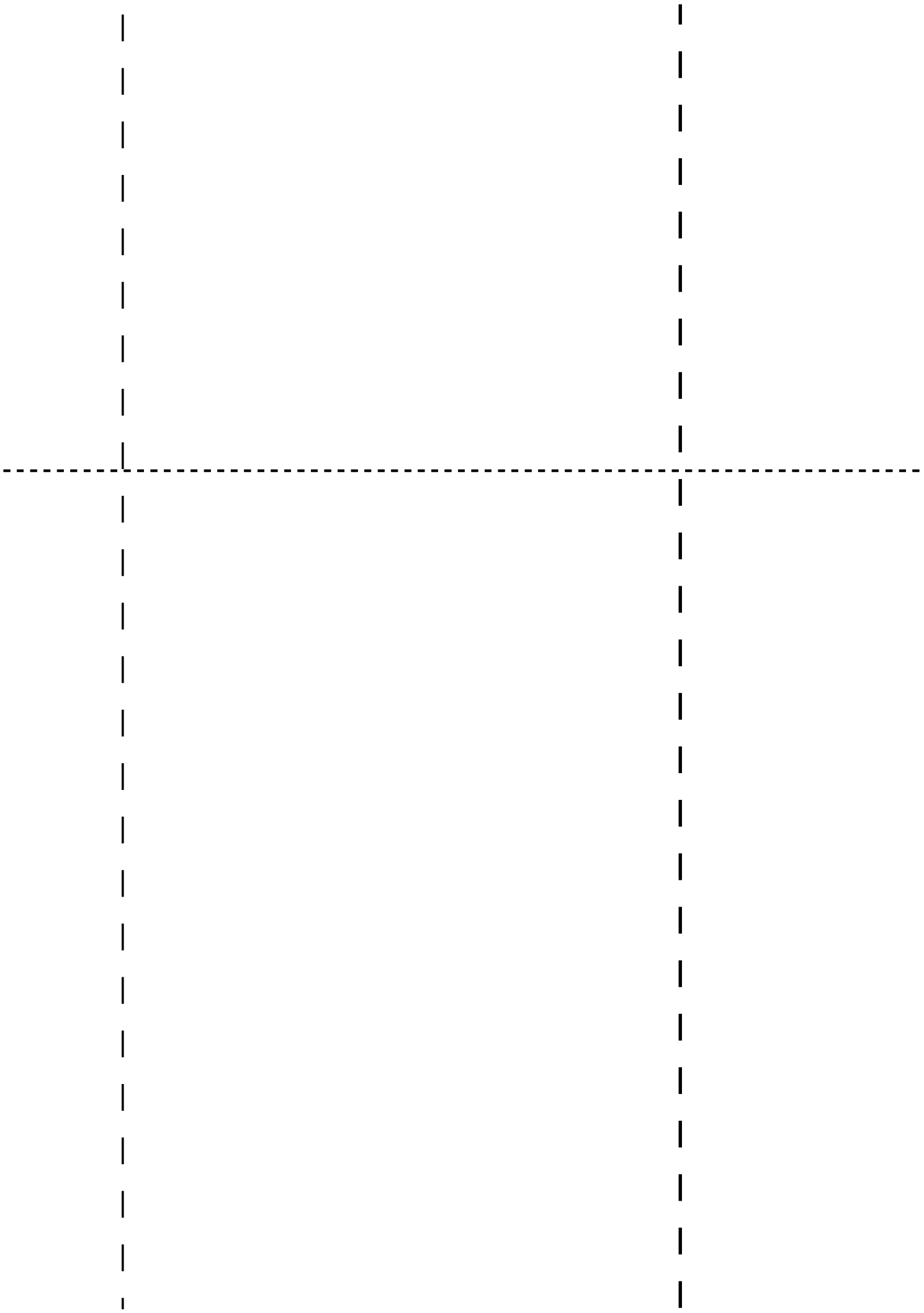


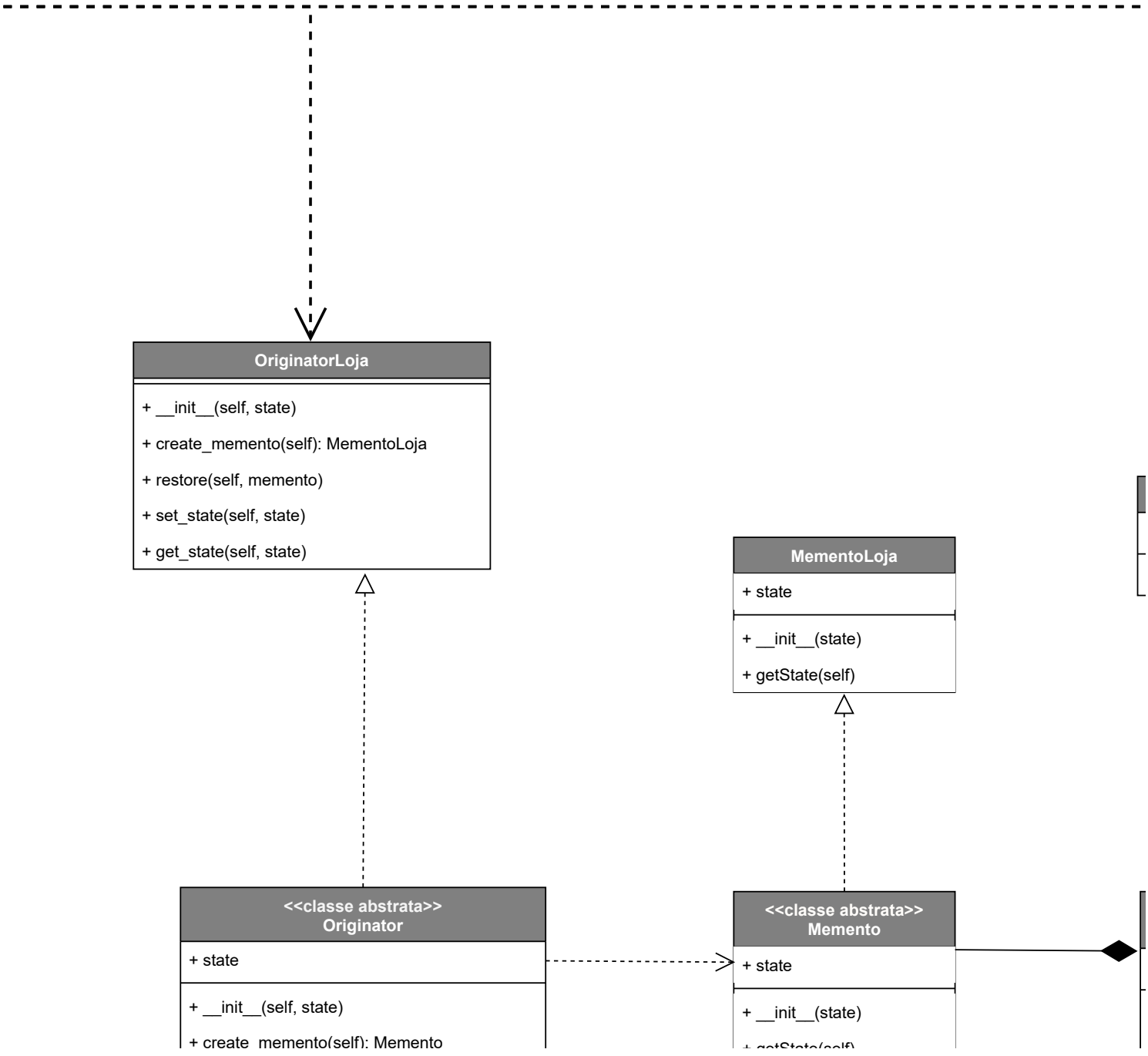

DatabaseAdapterNotificatorApi
+ __init__(self)
+ send(self, notificacao:Notificacao)
+ receive(self, id_loja:int)
FirebaseAdapterNotificatorApi
+ __init__(self)
+ send(self, notificacao:Notificacao)
+ receive(self, id_loja:int)
<<classe abstrata>>
NotificatorApi
+ send(self, notificacao:Notificacao)
+ receive(self, id_loja:int)
abc

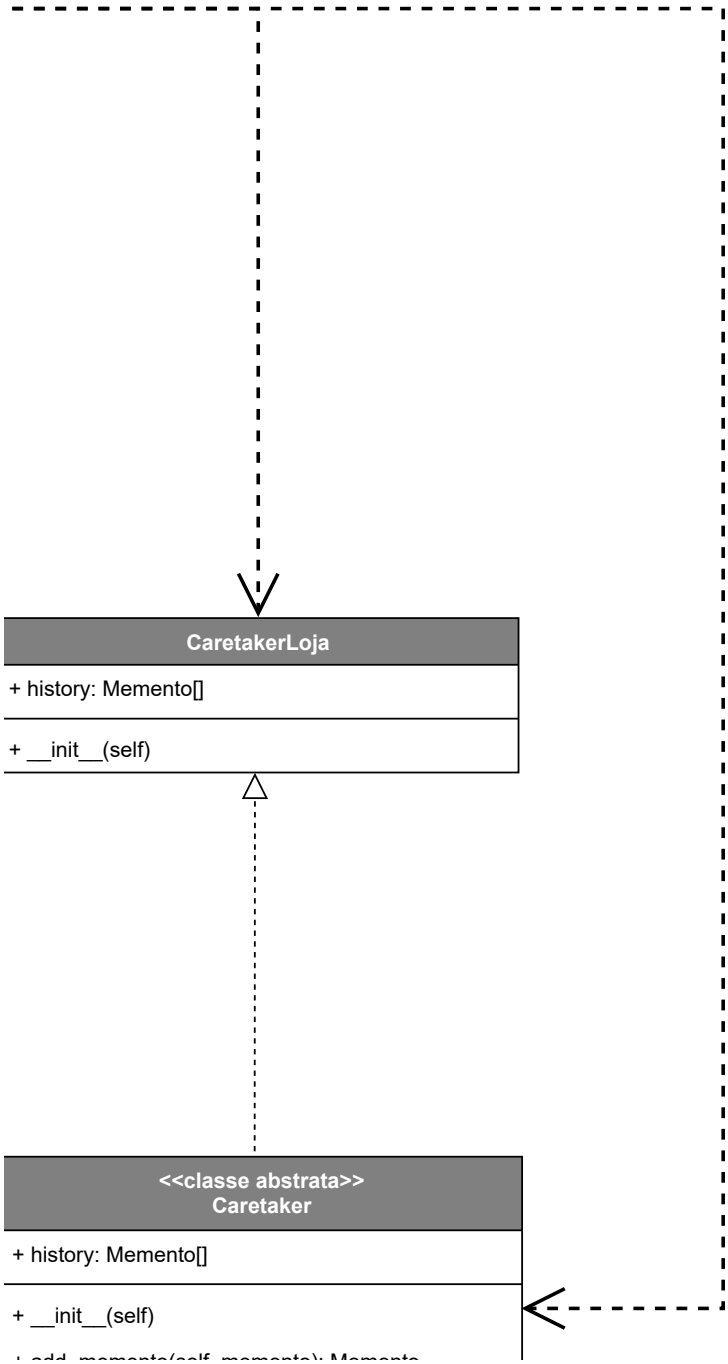

Exception
NotificationError
+ __init__(message: str):NotificationError
NotificatorApiSingleton
- notificator: NotificatorApi
+ build_send_notificacao(self, loja:Loja, id_usuario:int): void
+ get_notificacoes(self, id_loja): void


comandos
GerenciadorComandos
comando_autenticar_usuario(autenticador, username: str, senha: str): Usuario
comando_adicionar_usuario(gerenciador_usuarios, usuario)
comando_adicionar_loja(gerenciador_lojas, loja)
comando_adicionar_produto(gerenciador_produtos, produtos)
comando_editar_usuario(gerenciador_usuarios, id_usuario: int, usuario)
comando_editar_loja(gerenciador_lojas, id_loja: int, loja, id_usuario:int)
comando_editar_produto(gerenciador_produtos, id_produto: int, produto)
comando_excluir_usuario(gerenciador_usuarios, id_usuario: int)
comando_excluir_loja(gerenciador_lojas, id_loja: int)
comando_excluir_produto(gerenciador_produtos, id_produto: int)
comando_enviar_notificacao(loja, id_usuario:int)
comando_listar_notificacoes(loja, id_loja)


Fachada
+ gerenciador_usuarios: GerenciadorUsuarios
+ gerenciador_lojas: GerenciadorLoja
+ autenticador: AutenticacaoUsuario
+ usuario_autenticado: Usuario
+ __init__(tipo_persistencia: str): Fachada
+ login(): void
+ inicial_administrador(): void
+ inicial_gerente(): void
+ inicial_vendedor(): void
+ gerenciar_usuarios_como_administrador(): void
+ adicionar_usuario_tela_administrador(): void
+ adicionar_usuario_tela_gerente(): void
+ adicionar_administrador(): void
+ adicionar_usuario_como_administrador(tipo_usuario: str): void
+ adicionar_usuario_como_gerente(tipo_usuario: str): void
+ associar_loja_tela_inicial(): int
+ associar_loja(): int
+ adicionar_loja(): int
+ listar_tela_administrador(): void
+ listar_tela_gerente(): void
+ listar_administradores(): void
+ listar_usuarios(id_loja: int): void
+ filtrar_usuarios_por_loja(): void
+ listar_lojas(): void
+ listar_lojas_fluxo_filtro(): void
+ pesquisar_entidade(dados: dict, tipo_entidade: str): void
+ pesquisar_loja_fluxo_filtro(dados: dict): void
+ visualizar_usuario(id_: int): void
+ visualizar_loja(id_: int): void
+ visualizar_produto(id_: int): void
+ editar_usuario(informacoes: dict): void
+ editar_usuario_nome(nome: str): str
+ editar_usuario_username(username: str): str




OriginatorLoja
+ __init__(self, state)
+ create_memento(self): MementoLoja
+ restore(self, memento)
+ set_state(self, state)
+ get_state(self, state)
MementoLoja
+ state
+ __init__(state)
+ getState(self)
<<classe abstrata>>
Originator
+ state
+ __init__(self, state)
+ create_memento(self): Memento
<<classe abstrata>>
Memento
+ state
+ __init__(state)


CaretakerLoja
+ history: Memento[]
+ __init__(self)
<<classe abstrata>>
Caretaker
+ history: Memento[]
+ __init__(self)

+ editar_usuario_email(email: str):str

+ editar_usuario_confirmacao(informacoes: dict):bool

+ editar_usuario_descartar(): bool

+ excluir_usuario(): bool

+ editar_loja(caretaker_loja: CaretakerLoja, originator_loja: OriginatorLoja, id_usuario: int):void

+ editar_loja_nome(nome: str):str

+ editar_loja_endereco(endereco: str): str

+ editar_loja_confirmacao(informacoes: dict):bool

+ editar_loja_descartar(): bool

+ excluir_loja():bool

+ adicionar_produto(id_loja: int): void

+ editar_produto(informacoes: dict): void

+ editar_produto_nome(nome: str): str

+ editar_produto_descricao(descricao: str): str

+ editar_produto_preco(preco: str): str

+ editar_produto_quantidade(quantidade: str): str

+ editar_produto_confirmacao(informacoes: dict): bool

+ editar_produto_descartar(): bool

+ excluir_produto(): bool

+ listar_produtos(id_loja): void

+ opcoes_relatorios(): void

+ gerenciar_vendas(id_loja: int): void

+ realizar_venda(id_loja: int): void

+ definir_id_produto(repositorio: dict): int

+ definir_quantidade_produto(quant_disponivel: int): int

+ realizar_venda_confirmacao(informacoes: dict): bool

+ visualizar_notificacoes(): void

+ gerenciar_produtos(id_loja: int): void

+ gerenciar_lojas(): void

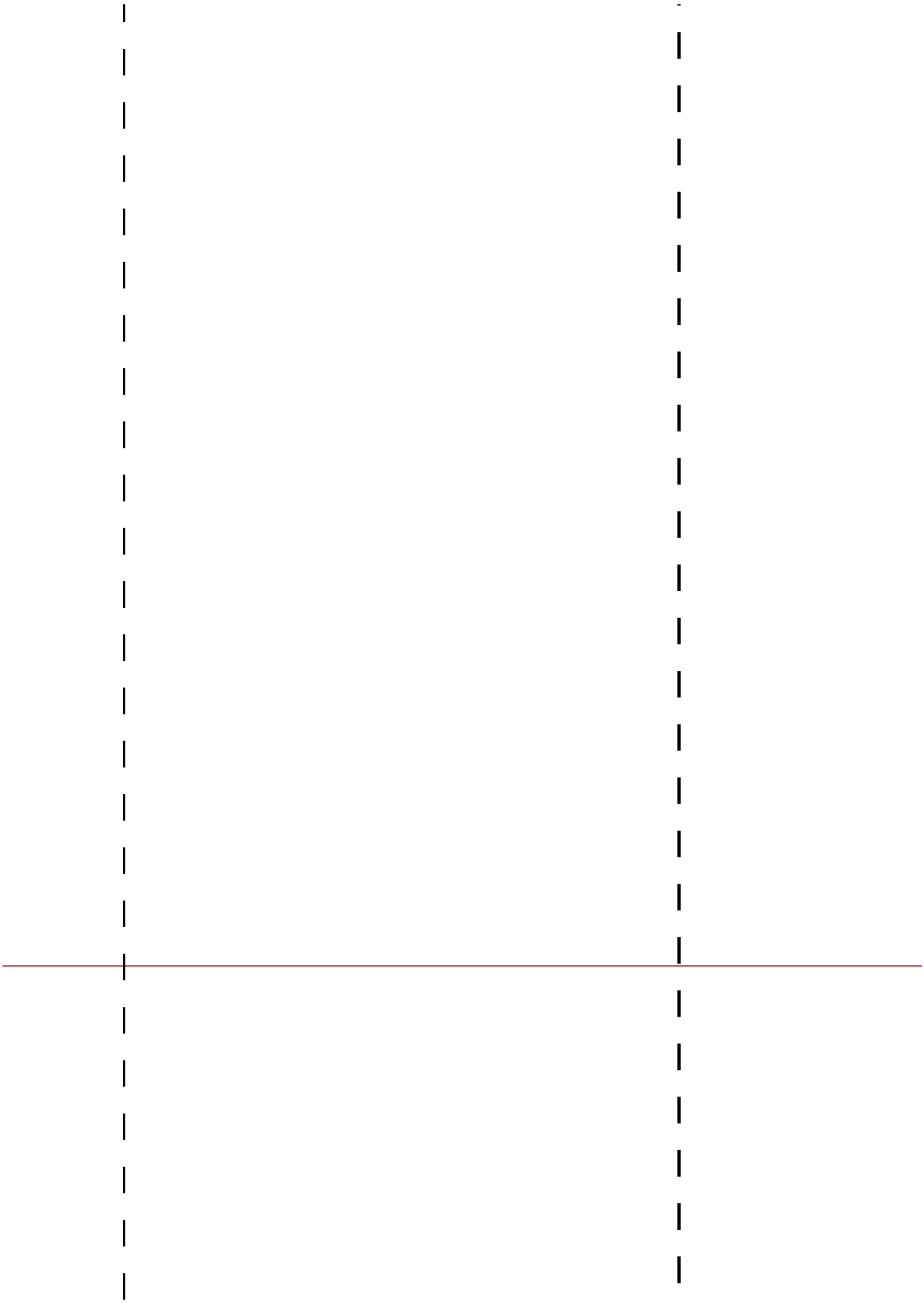
+ gerenciar_funcionarios(): void

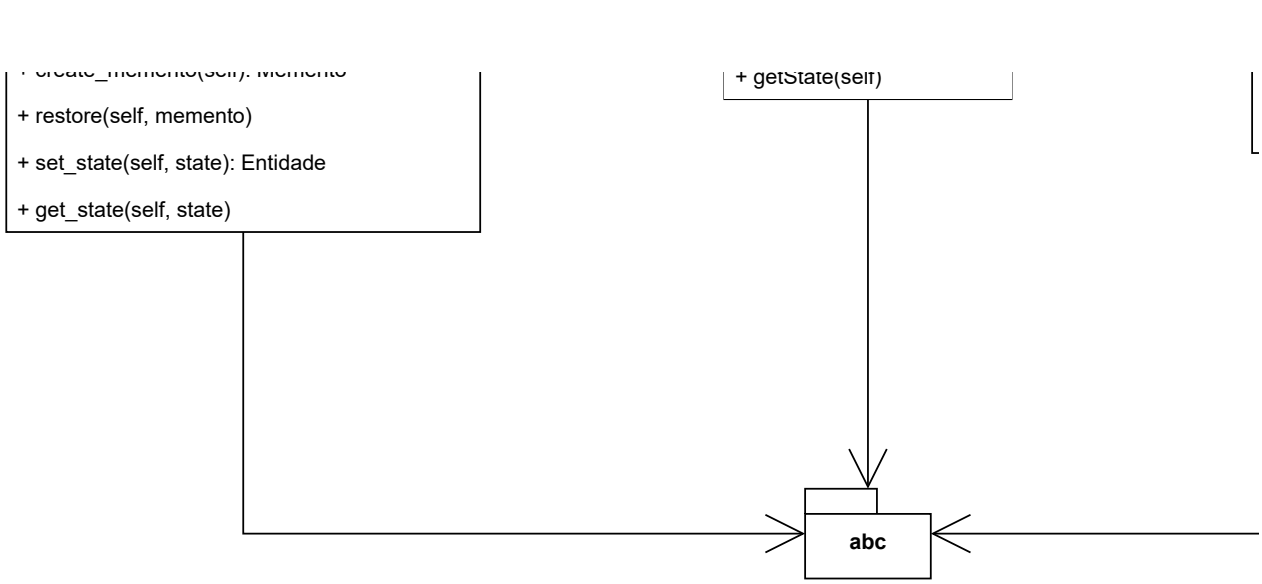
TemplateTela

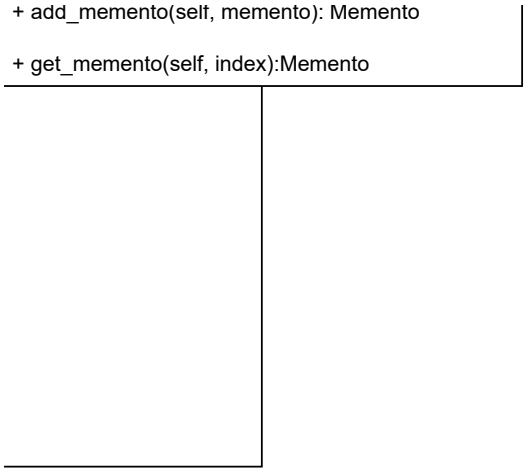
+ tela(informacoes = None): dict

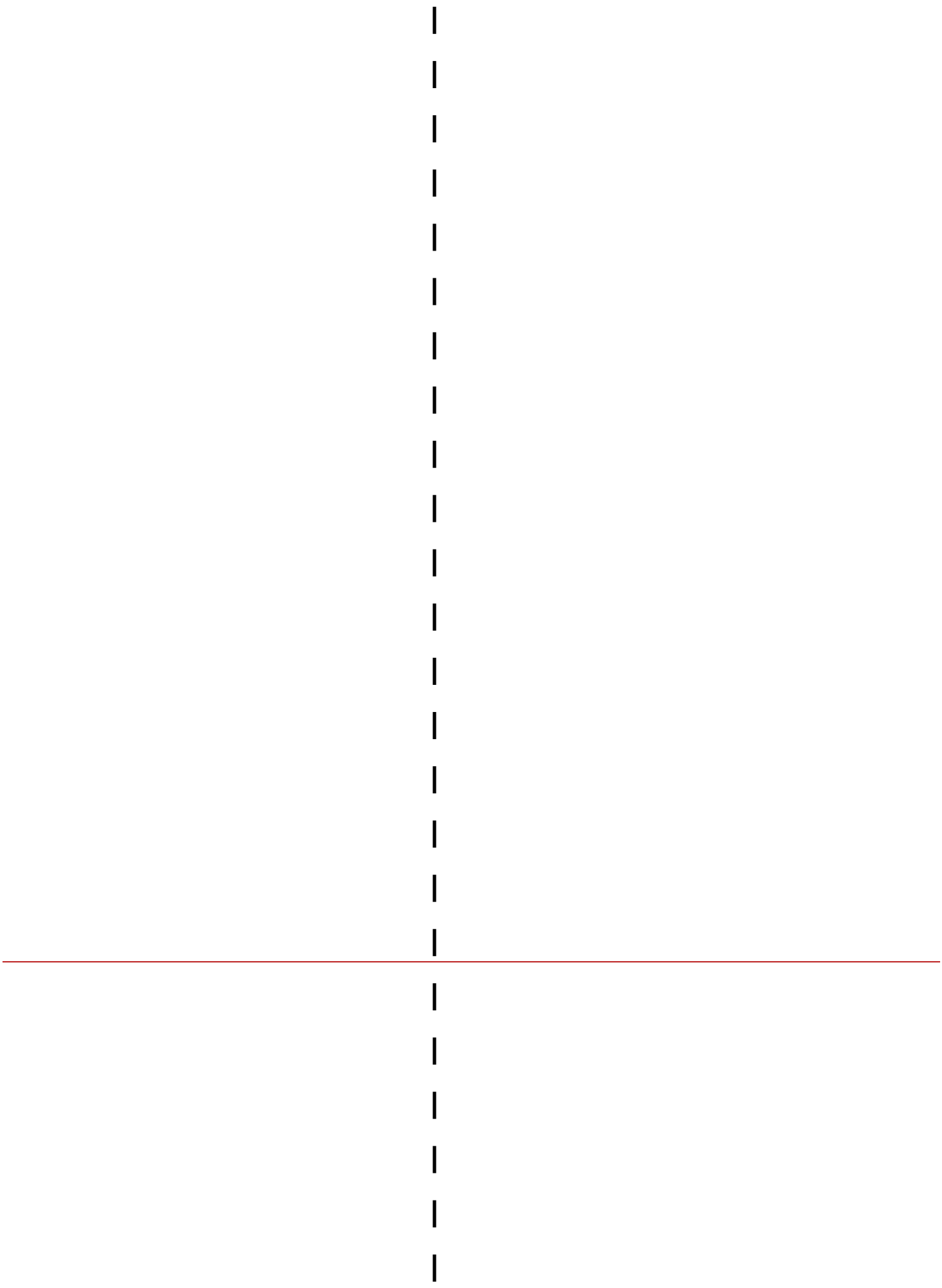
+ _titulo(): void

+ _listar_informacoes(informacoes: dict = None): void

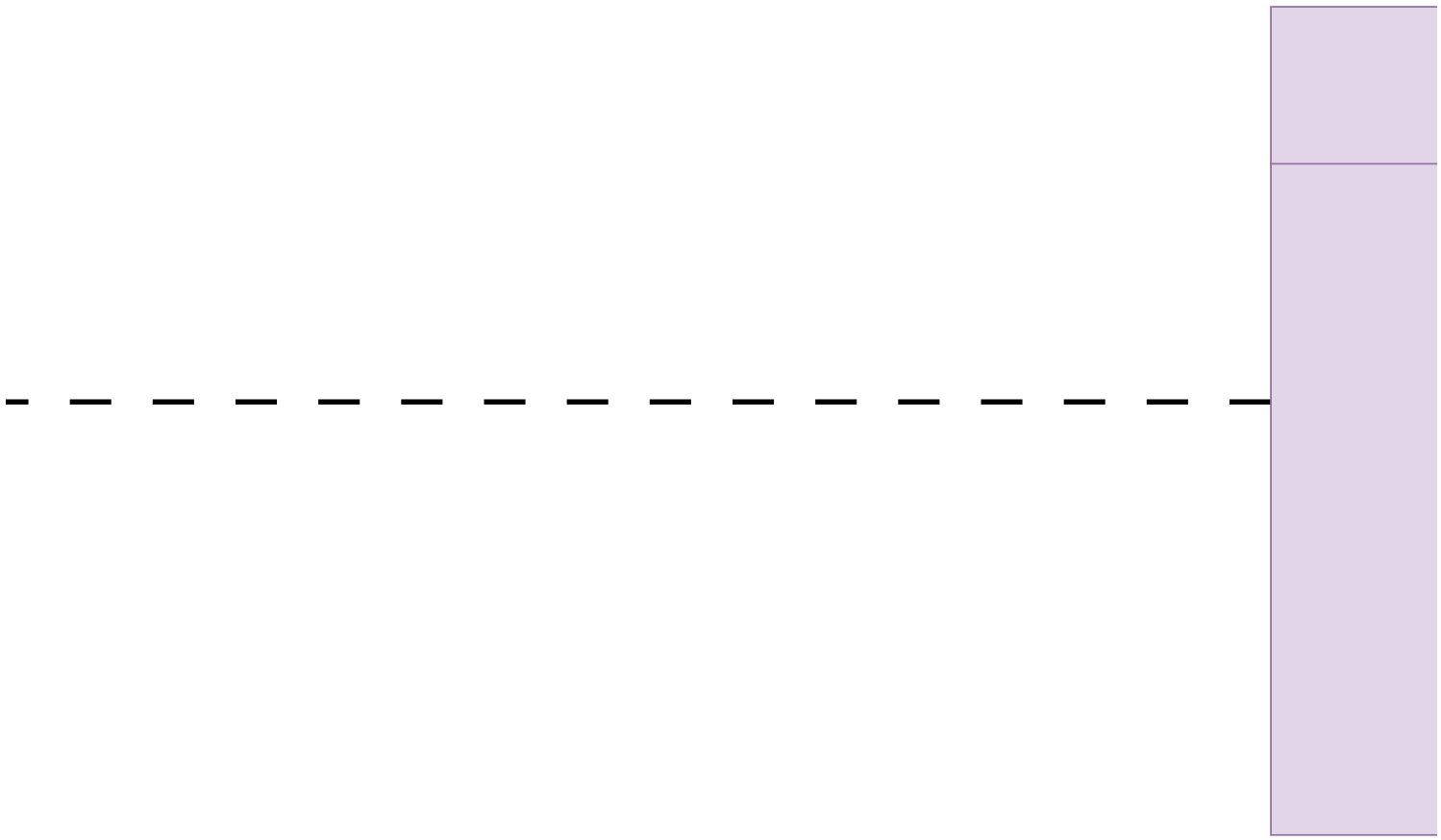


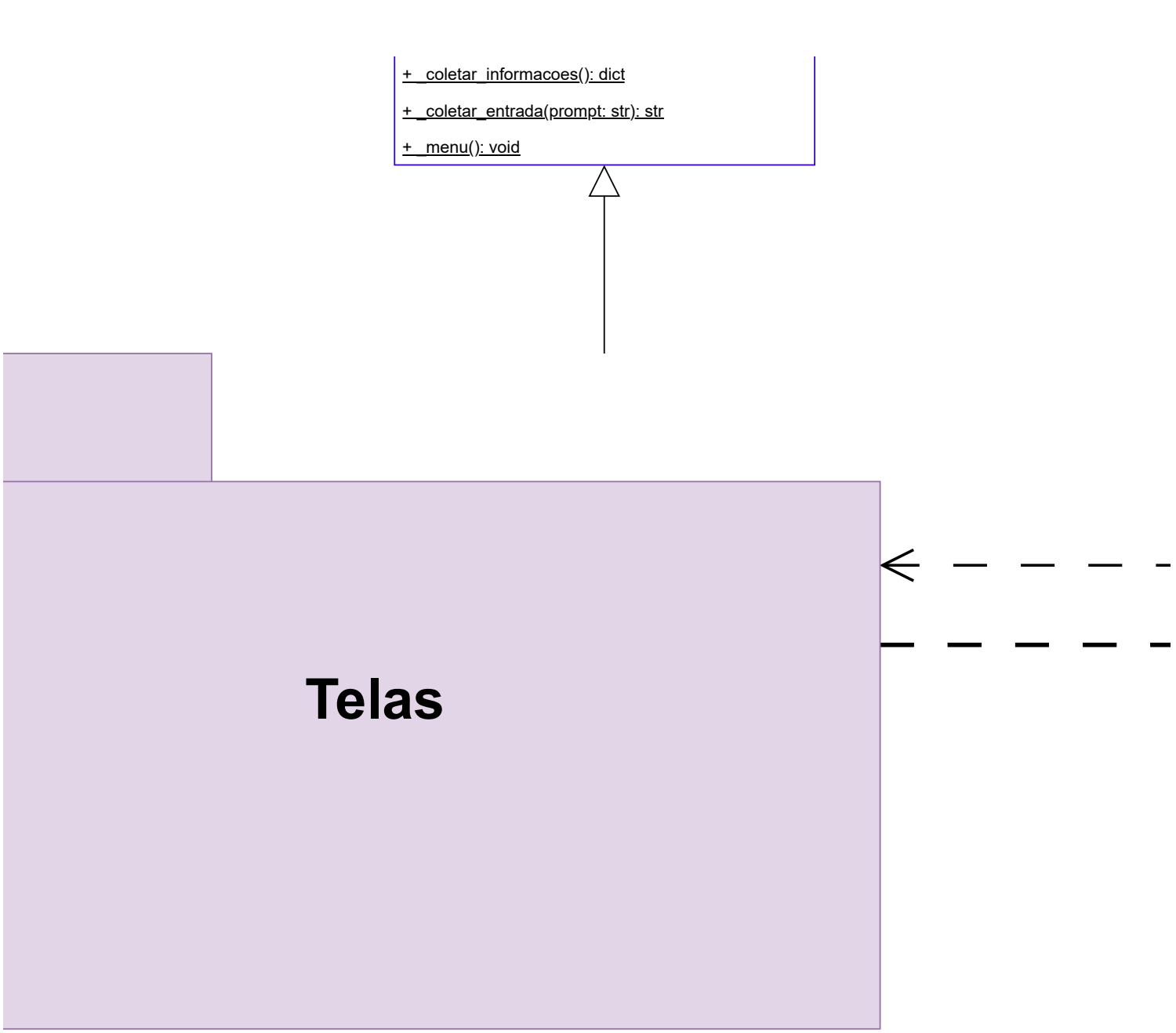

+ restore(self, memento)
+ set_state(self, state): Entidade
+ get_state(self, state)
+ getState(self)
abc


+ add_memento(self, memento): Memento
+ get_memento(self, index):Memento



Camada de apresentação




+ _coletar_informacoes(): dict
+ _coletar_entrada(prompt: str): str
+ _menu(): void
Telas