FATEC ANTÔNIO RUSSO

Engenharia de Software III – 4º Semestre – Matutino

ENGENHARIA DE SOFTWARE III : LISTA I

Bruno Cavalcante, RA 1680481611008

Henrique Borsatto de Campos, RA 1680481611020

Juliana Cardoso Malton, RA 1680481611003

Renan Dell'Monica Assoni, RA 1680481611027

Victor Zampieri Marinho, RA 1680481612023

São Caetano do Sul (SP)

setembro/2017

SUMÁRIO

1. INFORMAÇÕES

- Todas as implementações de código foram escritas em C#. Para adequar-se a tal, para os tipos *Date* do diagrama foi utilizada a classe *DateTime*, e para os tipos *List* foi utilizada a interface *IList*.
- Foi utilizada a sintaxe de Propriedades Automáticas do C#, onde a keyword *public* se refere à visibilidade dos getters e setters gerados pelo compilador a partir da sintaxe {*get; set;}*, e os atributos gerados são sempre privados conforme especificação da linguagem C#.
- No exercício de Gen/Espec foi utilizada keyword *virtual* para que não haja necessidade de fazer cast para que o método correto seja invocado (no Java, todos os métodos são *virtual* por padrão, já no C#, as chamadas por padrão são determinadas em tempo de compilação, e para mudar isso é necessário o uso da keyword *virtual*).
- Os getters e setters foram abreviados por questões de clareza e simplicidade dos diagramas, o tipo de retorno dos getters não é *void*¸e sim o tipo de cada um dos atributos.
- Alguns diagramas ficaram espalhados e grandes, não sendo legíveis na folha. Para ler, a função de zoom funciona pois o tamanho original da imagem foi preservado, apesar dela estar encolhida na folha.

2. CASOS DE USO TEXTUAIS

2.1. CSU01 – Gestão de Visita

Descrição: Permitir que o fluxo de entrada e saída do museu seja controlado através de um cartão magnético. No sistema ficam disponíveis informações básicas sobre o visitante cadastrado.

Atores: Recepcionista, Visitante.

Fluxo Principal

1. O Visitante solicita a entrada para a Recepcionista.
2. A Recepcionista verifica se o Visitante já está cadastrado no sistema.
3. A Recepcionista emite o cartão magnético e o entrega para o Visitante. Caso de uso Emitir Cartão Magnético <<include>>
4. Visitante efetua pagamento.
5. O Visitante utiliza o cartão magnético na Catraca.
6. O Sistema da Catraca valida o cartão.
7. O Sistema da Catraca libera a passagem para o Visitante. Caso de uso Liberar Passagem <<extend>>
8. Ao sair, o visitante devolve o cartão para a Recepcionista.
9. Caso de uso é encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 3 - Visitante não cadastrado.

3.a. A Recepcionista informa dados do Visitante. Caso de uso Cadastrar Visitante <<extend>>.

- 3.1.a. Recepcionista informa Nome do Visitante.
- 3.2.a. Recepcionista informa CPF do Visitante.
- 3.3.a. Recepcionista informa Telefone do Visitante.
- 3.4.a. Recepcionista informa E-mail do Visitante.

3.b. A Recepcionista emite o cartão magnético e entrega para o Visitante.

3.c. Retorna para o Fluxo Principal no passo 4.

FA2 – Alternativa ao passo 3 - Visitante solicita atualização de seu cadastro.

3.a. A Recepcionista informa novos dados para atualizar o cadastro do visitante no caso de uso Cadastrar Visitante. <<extend>>.

3.1.a. Visitante informa qual dado ele deseja alterar (Telefone e/ou E-mail).

3.b. A Recepcionista emite cartão magnético e entrega para o Visitante.

3.c. Retorna para o Fluxo Principal no passo 4.

Fluxos de Exceção

FE1 **–** Exceção no passo 6 - Sistema da Catraca detecta que o cartão não é valido.

6.a. Sistema da Catraca não libera entrada do Visitante.

6.b. Retorna para o passo 1 do Fluxo Principal.

2.2. CSU02 – Gestão de Acervo

Descrição: Permitir gerenciar acervo de obras do museu, tanto as obras cadastradas no acervo físico quanto às obras cadastradas no acervo virtual. Também é possível realizar consulta de obras.

Atores: Curador

Fluxo Principal

1. Curador acessa a tela de gerência de acervo.
2. Curador informa dados da obra para cadastro, caso de uso gerenciar obras.
   2.1. Curador informa Nome da Obra.
   2.2. Curador informa Autor da Obra.
   2.3. Curador informa Data de Criação da Obra.
   2.4. Curador informa Classificação da Obra.
   2.5. Curador informa Local de Produção da Obra.
   2.6. Curador informa Descrição da Obra.
3. Curador seleciona o acervo.
4. Curador seleciona “Cadastrar”.
5. Curador conclui o cadastro da obra no acervo.
6. Caso de uso Gerenciar Acervo é encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 4 - Curador seleciona “Buscar”, caso de uso Verificar Acervo.

4.a. O Curador seleciona Acervo Físico ele pretende realizar a consulta. Caso de uso Verificar Acervo Físico <<extend>>

4.b. Curador insere Nome ou Código da Obra para consultar.

4.c. Curador conclui a consulta.

4.d. Caso de uso é encerrado.

FA2 – Alternativa ao passo 4 – Curador seleciona “Buscar”. Caso de uso Verificar Acervo.

4.a. O Curador seleciona Acervo Virtual ele pretende realizar a consulta. Caso de uso Verificar Acervo Físico <<extend>>

4.b. Curador insere Nome ou Código da Obra para consultar.

4.c. Curador conclui a consulta.

FA3 – Alternativa ao passo 4 - Curador seleciona “Editar”.

4.a. Curador realiza consulta para achar a obra que deseja atualizar. Caso de uso Gerenciar Obras. <<include>>.

4.b. Curador insere os novos dados da obra selecionada.

4.c. Curador conclui a atualização.

4.d. Caso de uso é encerrado.

Fluxos de Exceção

FE1 – Exceção no passo 2 - Curador insere dados inválidos da obra.

2.a. Curador recebe notificação informando que os dados são inválidos.

2.b. Retorna para passo 2 do Fluxo Principal.

2.3. CSU03 – Gestão de Exposição

Descrição: Permitir que o Curador possa gerenciar as exposições (Temporárias e Permanentes) e reservar salas para as exposições.

Atores: Curador

Fluxo Principal

1. Curador acessa a tela de gerência de exposição.
2. Curador informa dados da exposição para cadastro de exposição permanente. Caso de uso Gerenciar Exposição Permanente. <<extend>>
    a. Curador informa Nome da Exposição.
    b. Curador informa Descrição da Exposição.
    c. Curador informa Nome ou Código da Obra.
3. Curador gerencia Obras que estarão na Exposição. Caso de uso Gerenciar Obras em Exposição <<include>>
4. Curador reserva Sala para a Exposição. Caso de uso Reservar Sala<<include>>
5. Sistema preenche automaticamente a lotação máxima da sala.
6. Curador seleciona “Cadastrar”.
7. Curador conclui o cadastro da exposição.
8. Caso de uso é encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 6 - Curador seleciona “Buscar”.

6.a. O Curador seleciona a exposição para consulta. Caso de uso Consultar Exposição
6.b. Curador insere dados (Nome ou Código da Exposição) para pesquisa.
6.c. Sistema lista obra. Caso de uso Listar Obras <<extend>>
6.d. Curador conclui a consulta, caso de uso encerrado.

FA2 – Alternativa ao passo 6 - Curador seleciona “Editar”.

6.a. O Curador seleciona a exposição para consulta. Caso de uso Consultar Exposição.
6.b. Curador insere novos dados para a exposição.
6.c. Curador conclui a atualização.
6.d. Caso de uso é encerrado.

FA3 – Alternativa ao passo 2 – Curador Cadastra Exposição Temporária. Caso de uso Gerenciar Exposição Temporária <<extend>>

2.a Curador informa dados da exposição para cadastro de exposição permanente. Caso de uso Gerenciar Exposição Temporária. <<extend>>

2.a.1 Curador informa Nome da Exposição.

2.a.2 Curador informa Descrição da Exposição.

2.a.3. Curador informa Nome ou Código da Obra.

2.a.4. Curador informa “Exposição Temporária”.

2.a.5. Curador informa Data de Início da Exposição.

2.a.6. Curador informa Data de Fim da Exposição.

2.b. Retorna para o passo 3 do Fluxo Principal.

Fluxos de Exceção

FE1 – Exceção no passo 2 - Curador insere dados inválidos.

2.a. Curador recebe notificação informando que os dados são inválidos.

2.b. Retorna para passo 2 do Fluxo Principal.

2.4. CSU04 – Gestão de Restauração

Descrição: Curador solicita uma restauração e acompanha todo o processo e o Restaurador atende a solicitação e atualiza o status da restauração.

Atores: Curador, Restaurador.

Fluxo Principal

1. Curador acessa a tela de “Gerenciar Restaurações”.
2. Curador solicita restauração da abra. Caso de uso Solicitar Restauração<<extend>>
3. Curador informa dados sobre a obra que irá ser restaurada.
   3.1. Curador informa Descrição da Restauração.
   3.2. Curador informa Data de Início da Restauração.
   3.3. Curador informa Data de Fim da Restauração.
   3.4. Curador informa Valor da Restauração.
   3.5. Curador informa Status da Restauração.
   3.6. Curador informa Número do Crachá do Restaurador.
   3.7. Curador informa Código da Obra.
4. Sistema envia notificação para restaurador.
5. Restaurador atende restauração. Caso de uso Atender Restauração <<include>>
6. Restaurador realiza restauração da obra.
7. Restaurador atualiza status da restauração. Caso de uso Atualizar Status Restauração<<include>>
8. Restaurador finaliza restauração.
9. Curador recebe notificação de restauração finalizada.
10. Caso de uso é encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 2 – Curador solicita consultar status restauração. Caso de uso Consultar Status Restauração <<extend>>

2.a. Curador insere Código da Restauração.
2.b. Curador verifica o status da restauração.
2.c. Caso de uso é encerrado

Fluxos de Exceção

FE3 – Exceção no passo 3 - Curador realiza o cadastro de forma errada

3.a. Curador é notificado sobre a inconsistência dos dados.

3.b. Retorna para passo 3 do Fluxo Principal.

2.5. CSU05 – Gestão de Evento

Descrição: Curador organiza um evento e convida Visitantes já cadastrados no sistema.

Atores: Curador, Visitante.

Fluxo Principal

1. Curador acessa tela de Gerenciar Eventos.
2. Curador seleciona “Agendar Evento”.
3. Curador informa dados do evento.
   3.1. Curador informa Nome do Evento.
   3.2. Curador informa Descrição do Evento.
   3.3. Curador informa data de Início do Evento.
   3.4. Curador informa data de Fim do Evento.
   3.5. Curador informa Número do Crachá do Curador Responsável.
   3.6. Curador informa Sala que ocorrerá o Evento.
   3.7. Curador informa Texto do Convite do Evento.
4. Sistema preenche automaticamente lotação máxima de visitantes/convidados para o Evento.
5. Curador seleciona os convidados a partir dos Visitantes já cadastrados no museu.
6. Curador cadastra o evento.
7. Curador solicita que o sistema envie e-mails com os convites para os convidados selecionados.
8. Visitante confirma a presença no evento por e-mail. Caso de uso Enviar Convite <<extend>>
9. Caso de uso é encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 8 - Visitante não confirma a presença

8.a. Visitante não confirma a presença no evento.
8.b. Caso de uso é encerrado.

FA2 – Alternativa ao passo 2 - Curador seleciona “Editar Evento”.

2.a. O Curador seleciona o evento para consulta.
2.b. Curador insere Código ou Nome do evento para pesquisa.
2.c. Curador seleciona evento para editar.
2.d. Curador insere os novos dados do evento.

2.e. Curador conclui a atualização.

2.f. Caso de uso é encerrado.

FA3 – Alternativa ao passo 2 - Curador seleciona “Cancelar Evento”.

2.a. Curador seleciona o evento para consulta.

2.b. Curador insere dados para pesquisa.

2.c. Curador seleciona evento para cancelar.

2.d. Curador confirma cancelamento do evento.

2.e. Visitante recebe e-mail com aviso sobre o cancelamento do evento.

2.f. Caso de uso é encerrado.

Fluxos de Exceção

FE2 – Exceção no passo 3.6 - A sala já reservada.

3.6.a. Curador é notificado que a sala já está reservada.

3.6.b. Curador seleciona outra sala para o evento.

3.6.c. Retorna para o passo 3.6 do Fluxo Principal.

2.6. CSU06 – Gestão de Venda de Souvenir

Descrição: Garantir toda a gestão de vendas de souvenires na loja, sendo possível manipular o estoque, compra de novos produtos e relatórios de vendas que serão disponíveis para o Gerente de Vendas.

Atores: Vendedor, Visitante, Gerente de Vendas.

Fluxo Principal

1. Visitante escolhe produto na prateleira.
2. Visitante se dirige ao caixa com o produto.
3. Vendedor informa itens da compra.
   3.1. Vendedor informa Código do Produto.
   3.2. Vendedor informa Quantidade.
4. Vendedor Adiciona Item.
5. Vendedor Finaliza Compra. Caso de uso Atualizar Estoque <<include>>
6. Caso de uso encerrado.

Fluxos Alternativos

FA1 – Alternativa ao passo 1 - Vendedor ou Gerente cadastra Souvenir

1.a. Vendedor ou Gerente informam os dados do souvenir para cadastro.
   1.1.a. Vendedor ou Gerente informam Código do Produto.
   1.2.a. Vendedor ou Gerente informam Nome do Produto.
   1.3.a. Vendedor ou Gerente informam Descrição do Produto.
   1.4.a. Vendedor ou Gerente informam Preço de Custo.
   1.5.a. Vendedor ou Gerente informam Preço de Venda.

1.b. Vendedor ou Gerente selecionam Cadastrar.

1.b. Caso de uso encerrado.

FA2 – Alternativa ao passo 1- Gerente de Vendas solicita produtos ao fornecedor.

1.a. Gerente de Vendas solicita produtos ao fornecedor.

1.b. Fornecedor recebe pedido. Caso de uso Receber Pedido <<include>>

1.c. Fornecedor realiza entrega.

1.d. Gerente de Vendas atualiza estoque na tela de Gerenciar Estoque. Caso de uso Atualizar Estoque <<include>>

1.e. Caso de uso encerrado.

FA3 – Alternativa ao passo 1 - Vendedor solicita a consulta de estoque

1.a. Vendedor verifica se produto está disponível no estoque. Caso de uso Consultar Estoque <<extend>>
1.b. Vendedor insere Nome ou Código do Produto no sistema para consulta.
1.c. Caso de uso encerrado.

FA4 – Alternativa ao passo 1 - Gerente de vendas solicita relatório

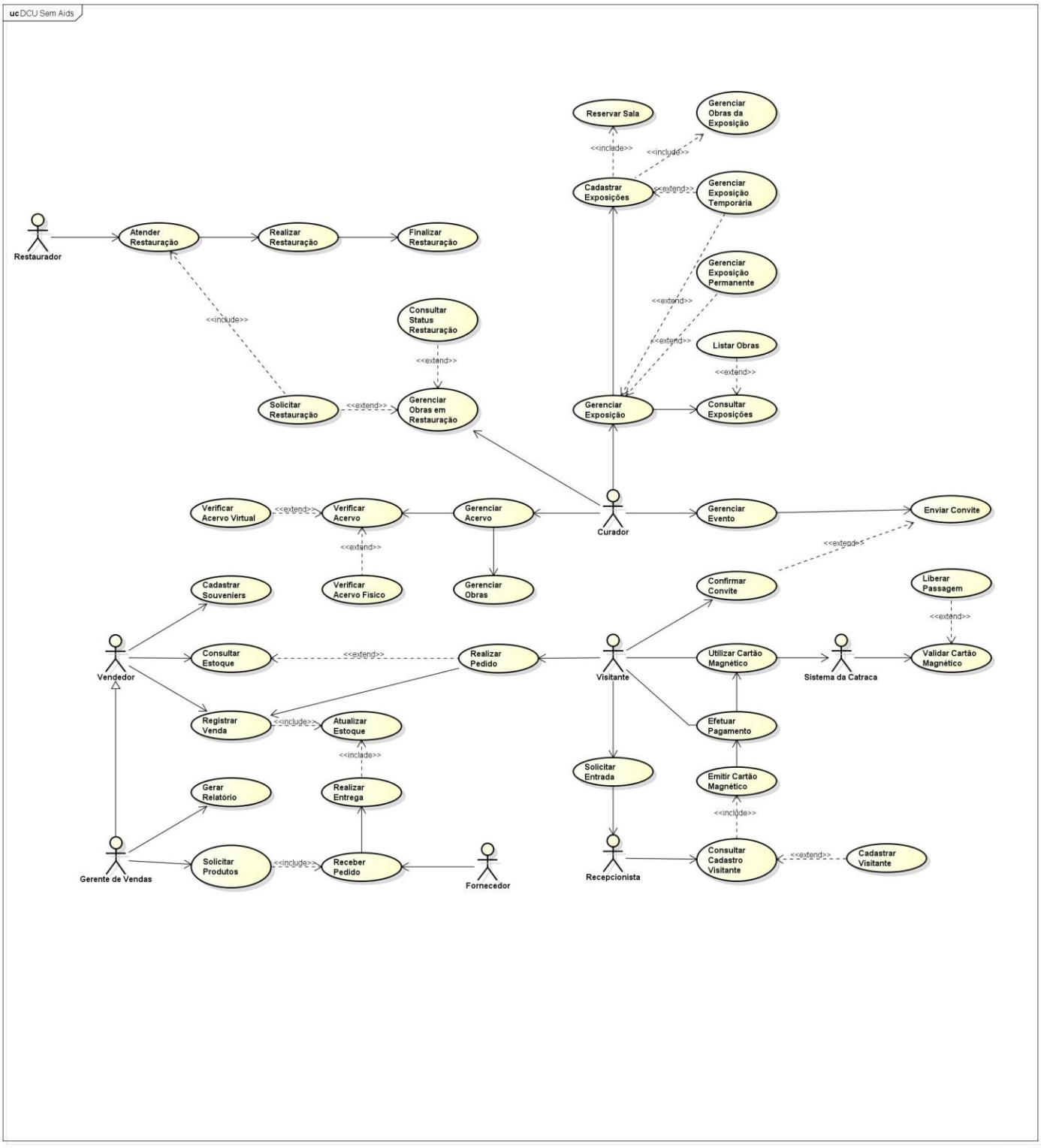
1.a. Gerente de vendas solicita geração de relatório de vendas.
1.b. Sistema gera relatório de vendas e exibe ao gerente.
1.c. Caso de uso encerrado.

Fluxos de Exceção

FE1 – Exceção no passo 2 - Produto com defeito

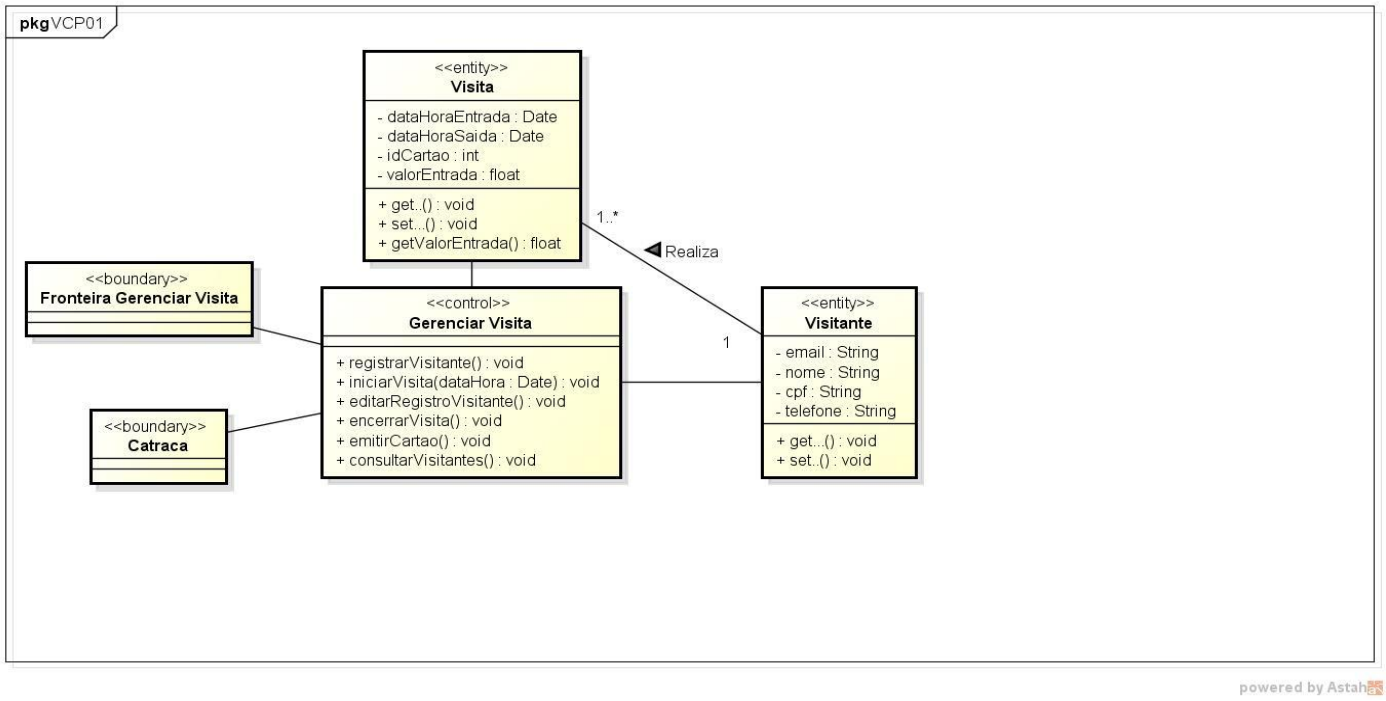
2.a. Visitante mostra ao Vendedor que o produto está com defeito.
2.b. Vendedor registra troca de produto no sistema na tela de Vendas.
2.c. Sistema atualiza estoque. Caso de uso Atualizar Estoque <<include>>
2.c. Caso de uso encerrado.

3. DIAGRAMA DE CASOS DE USO

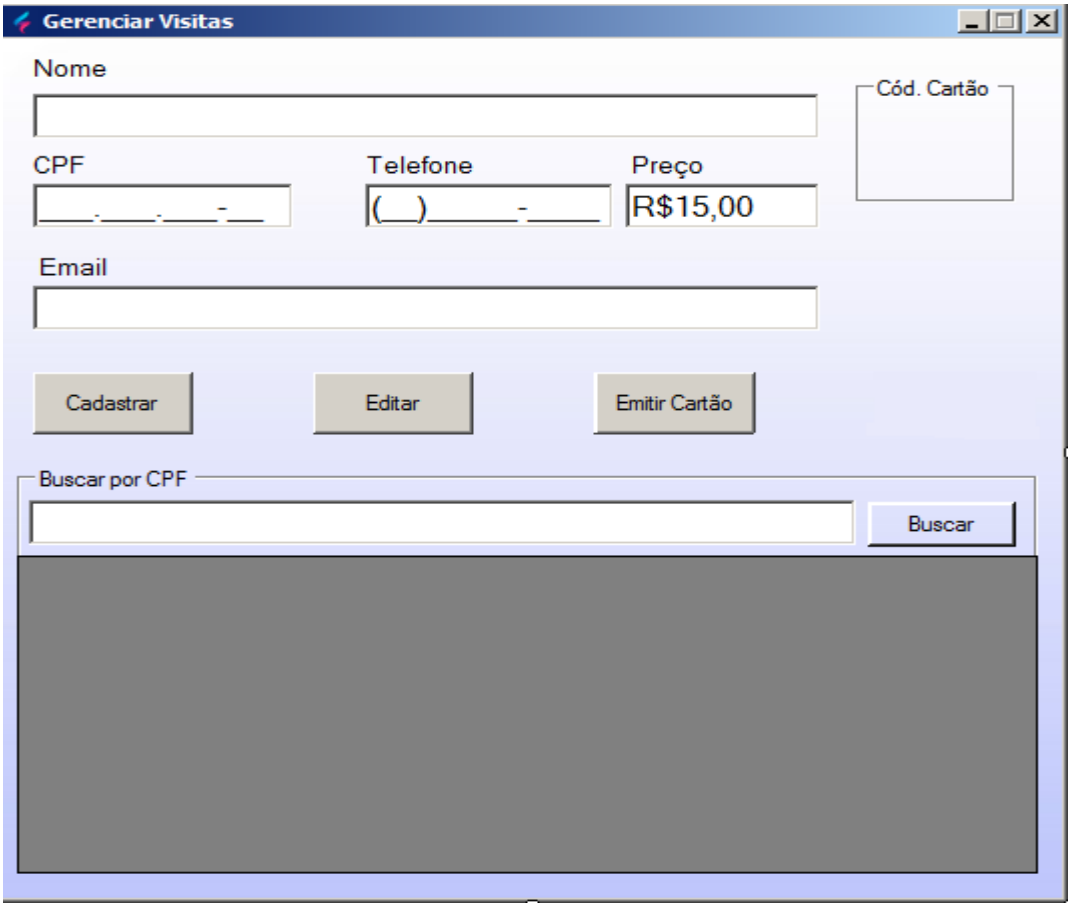


4. VISÕES DE CLASSES PARTICIPANTES

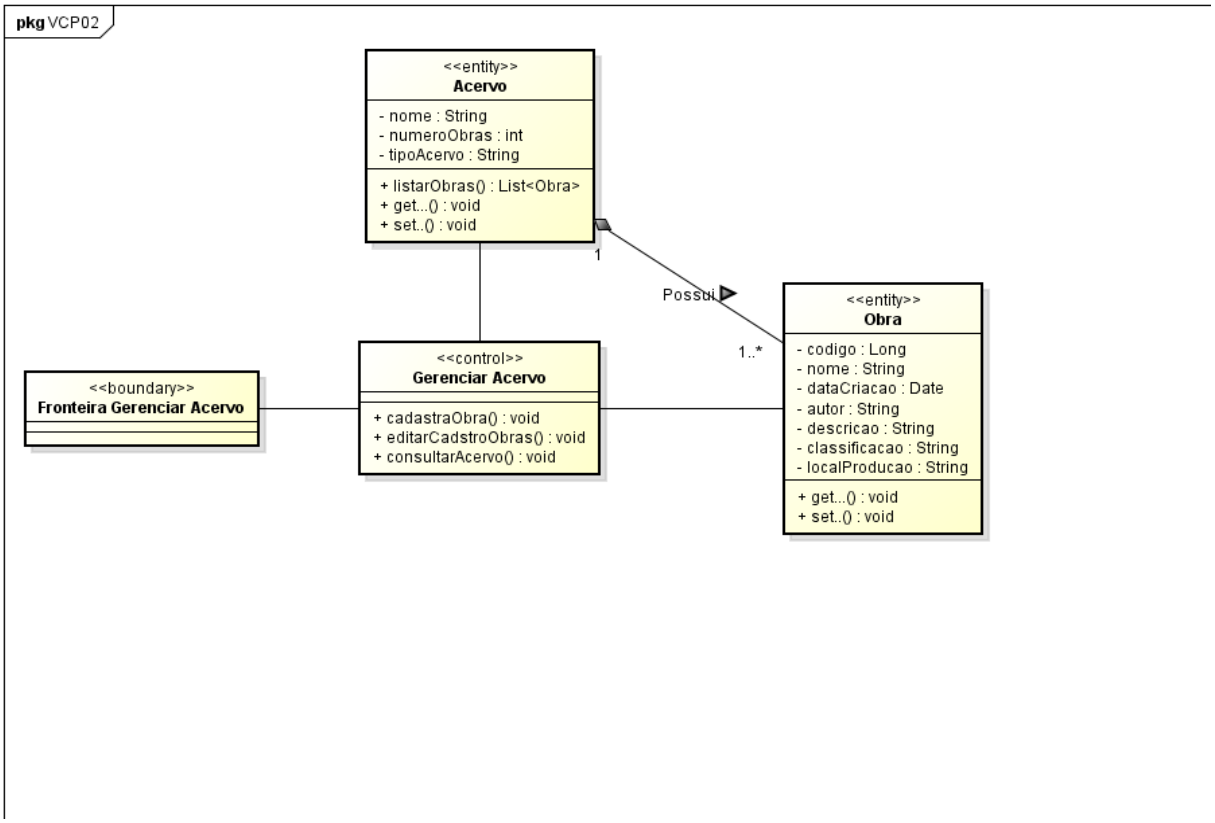
4.1. VCP – CSU01 – Gestão de Visita



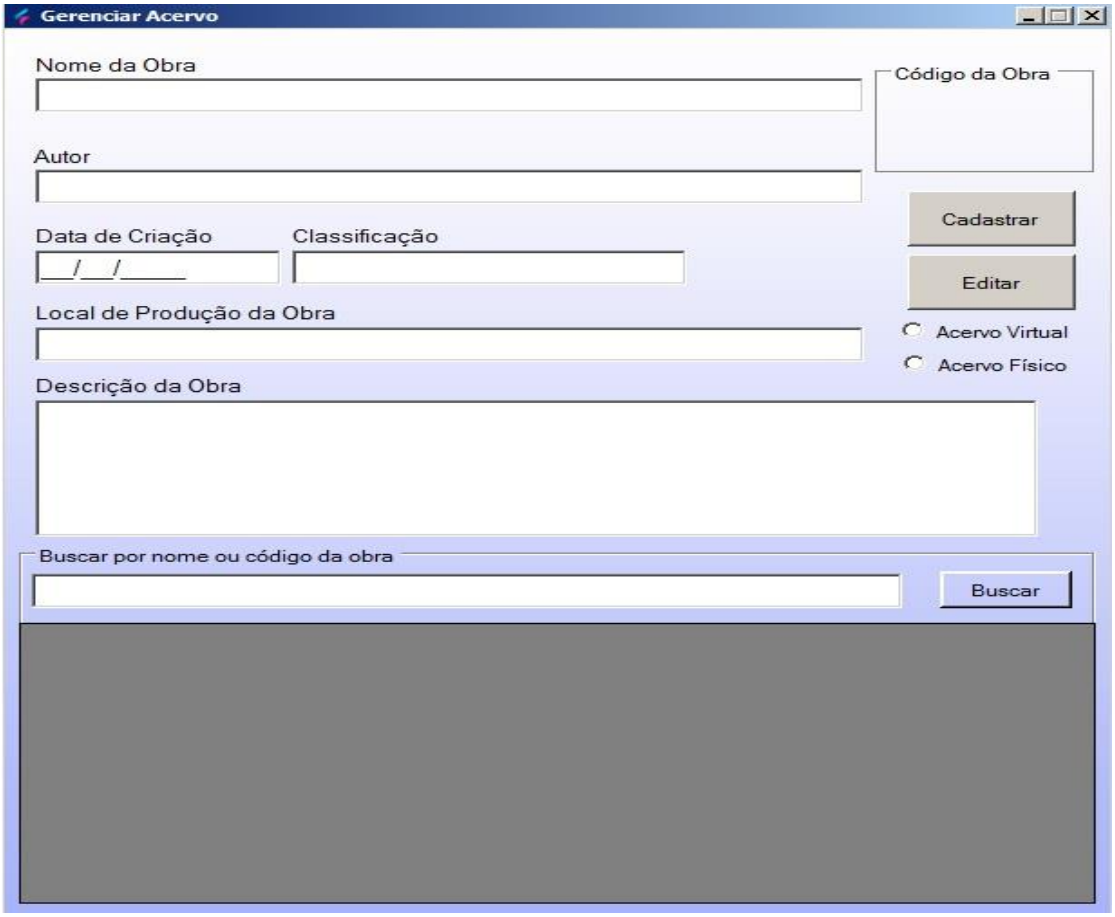
Protótipo de tela:



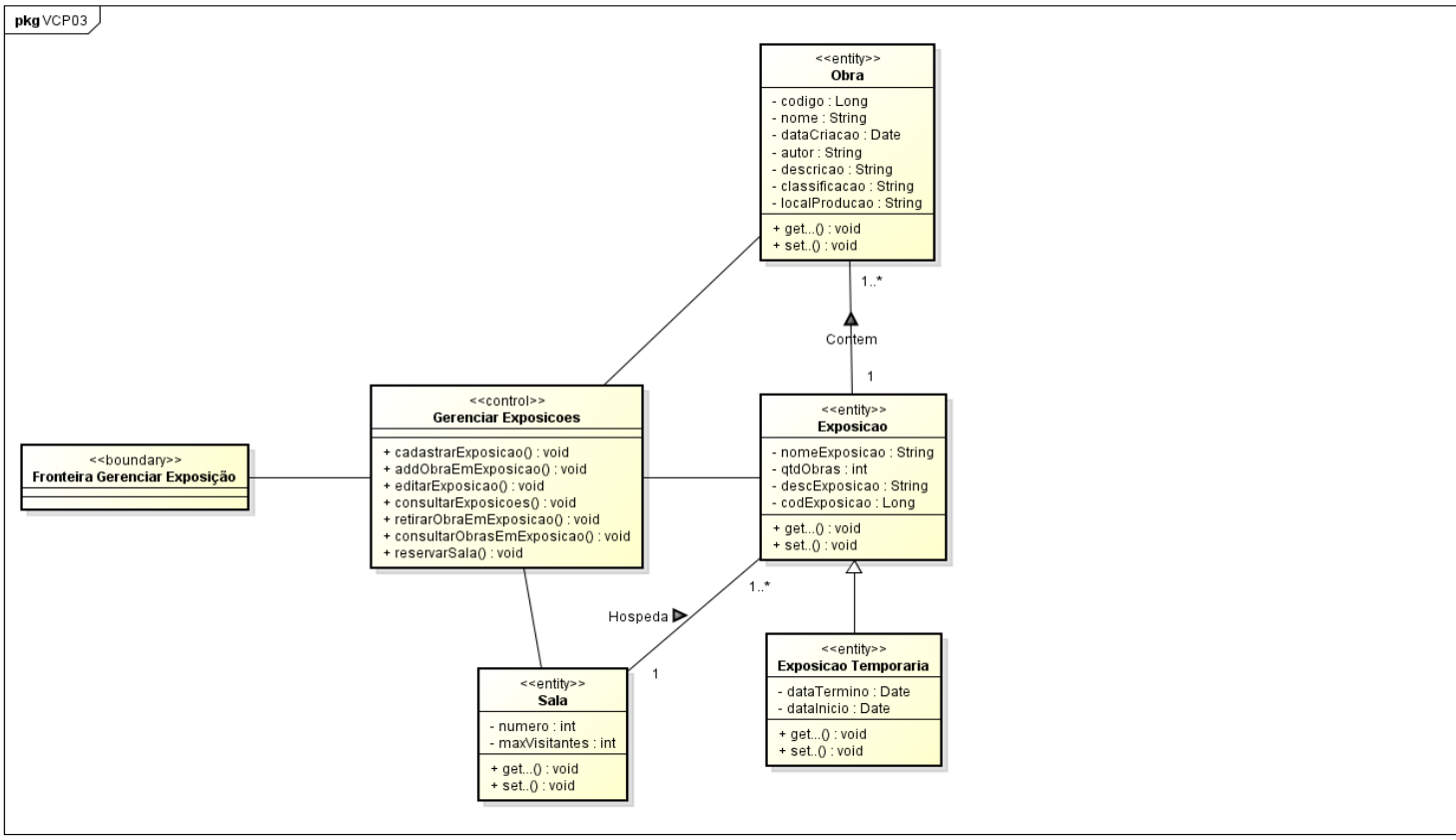
4.2. VCP – CSU02 – Gestão de Acervo



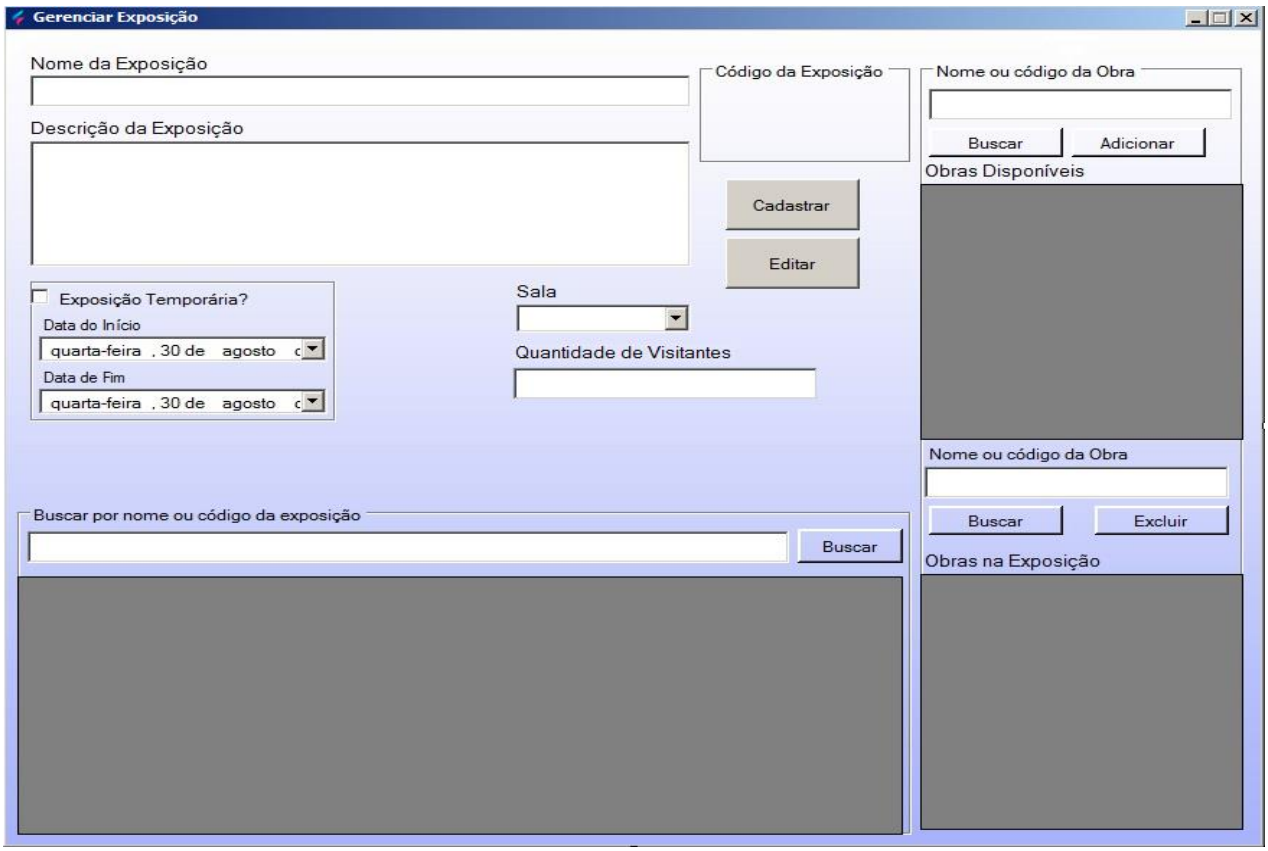
Protótipo de tela:



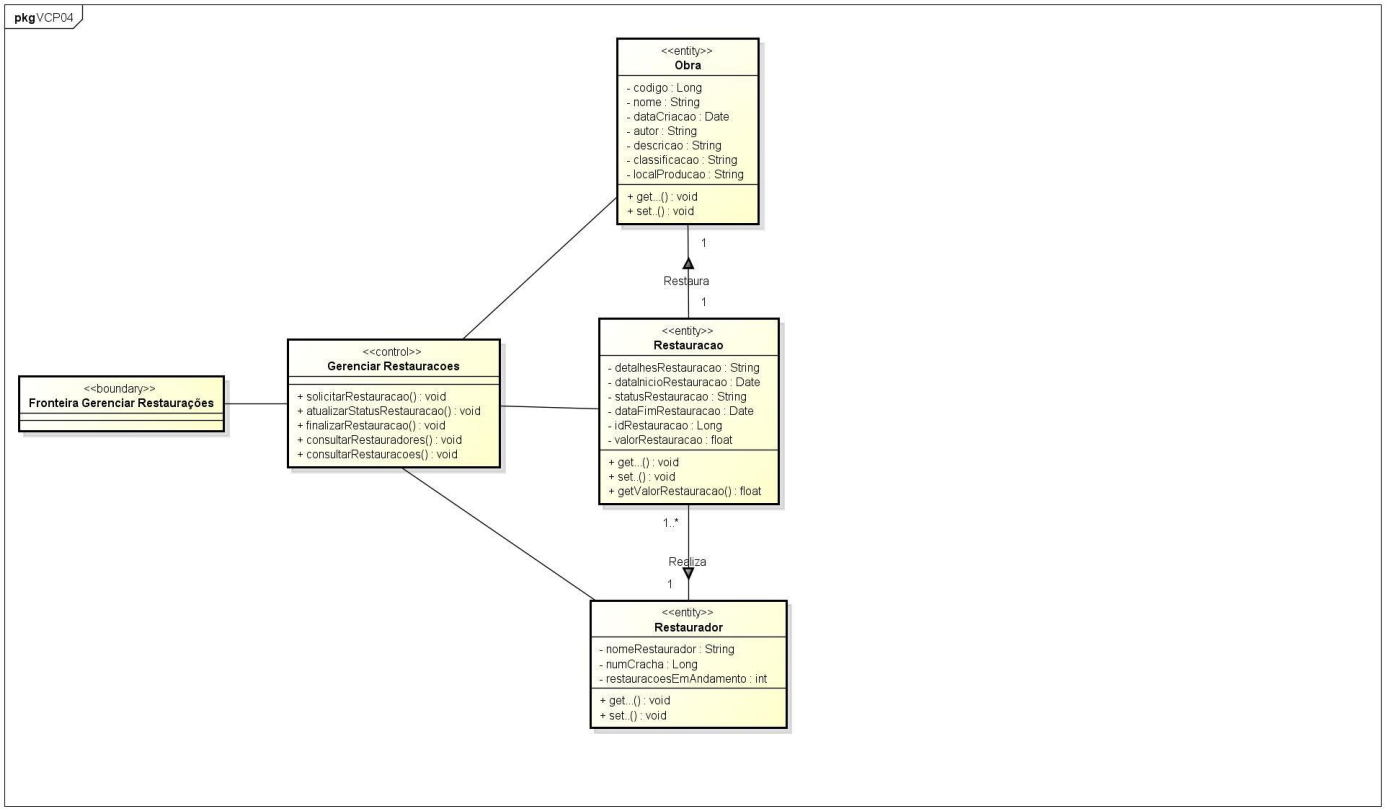
4.3. VCP – CSU03 – Gestão de Exposição



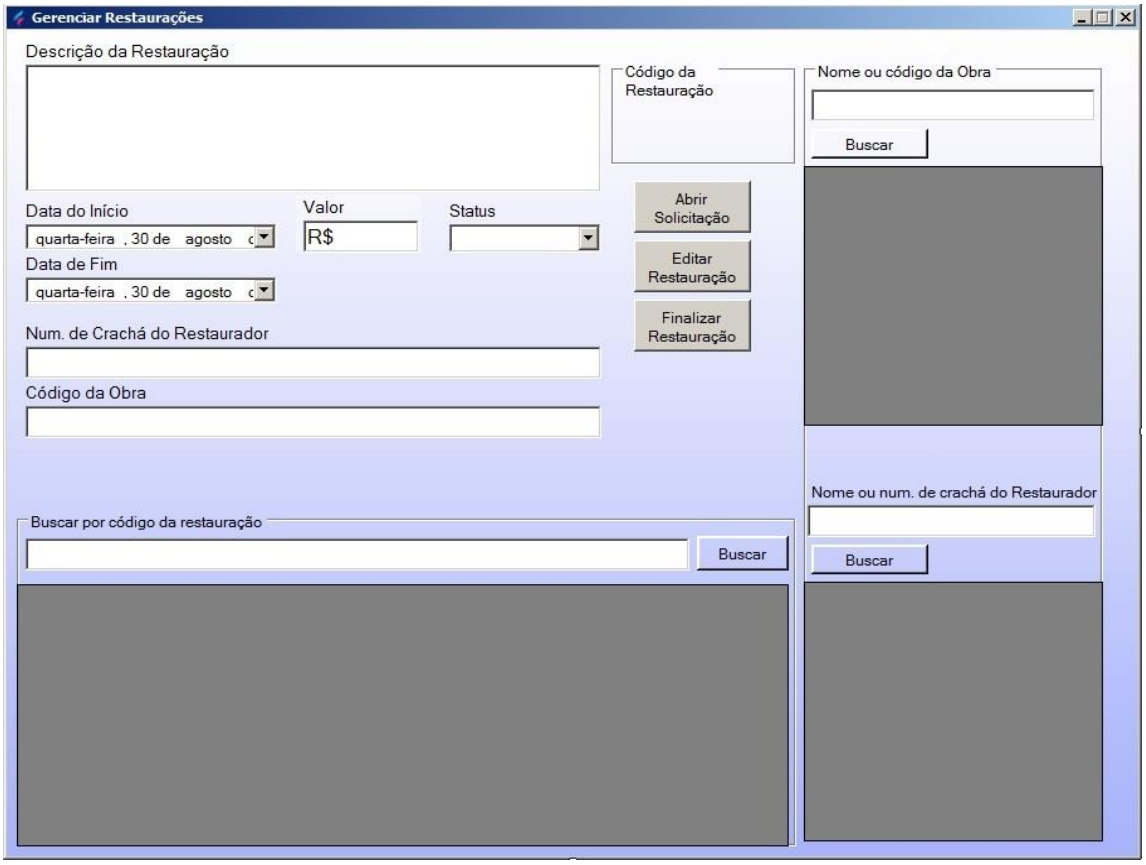
Protótipo de tela:



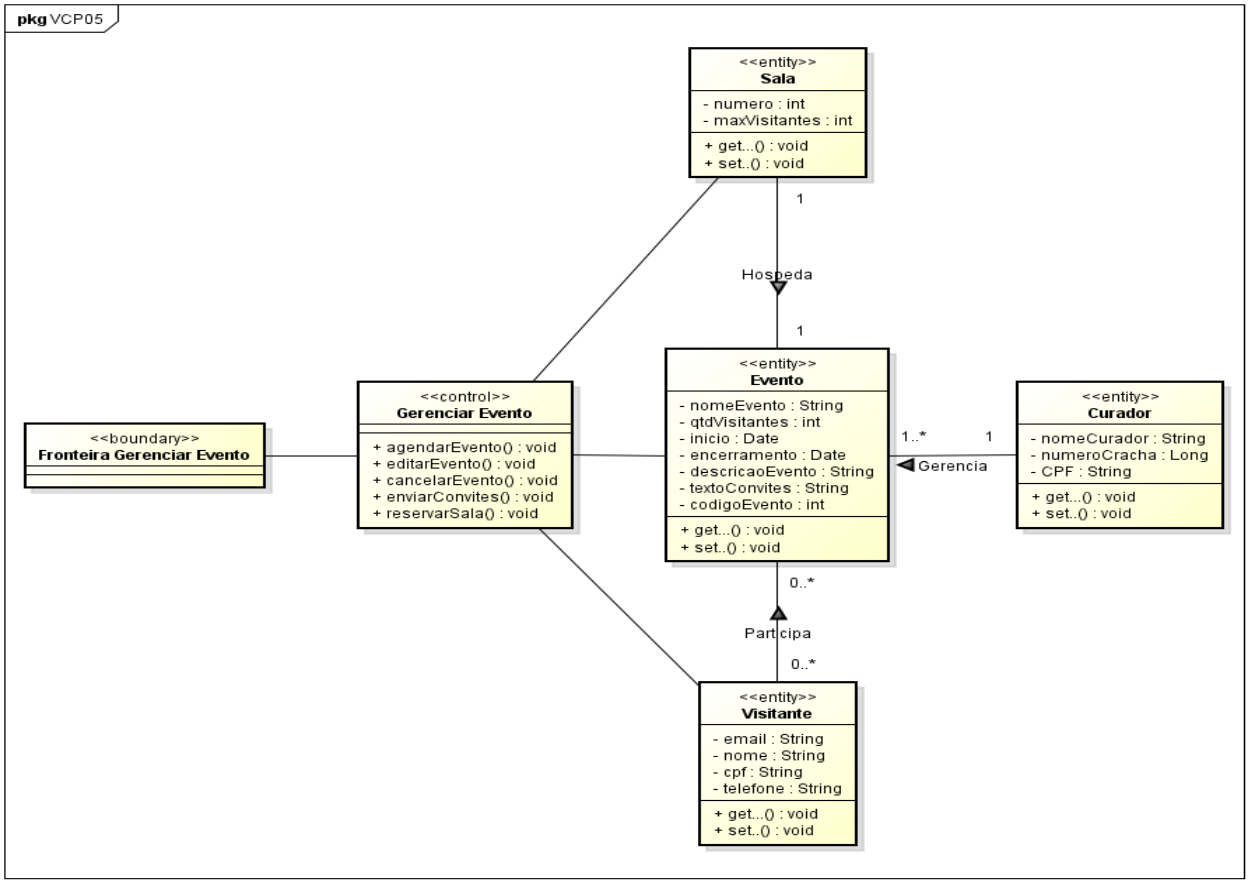
4.4. VCP – CSU04 – Gestão de Restauração



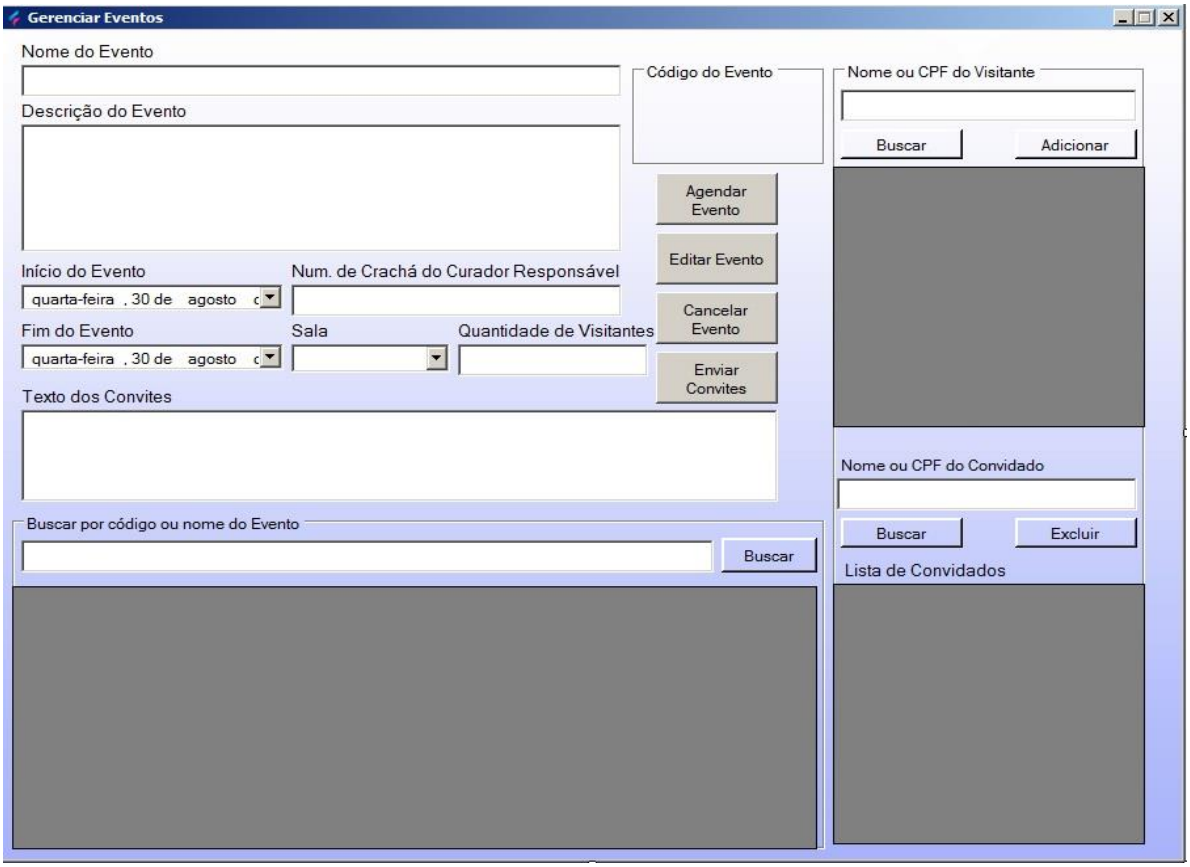
Protótipo de tela:



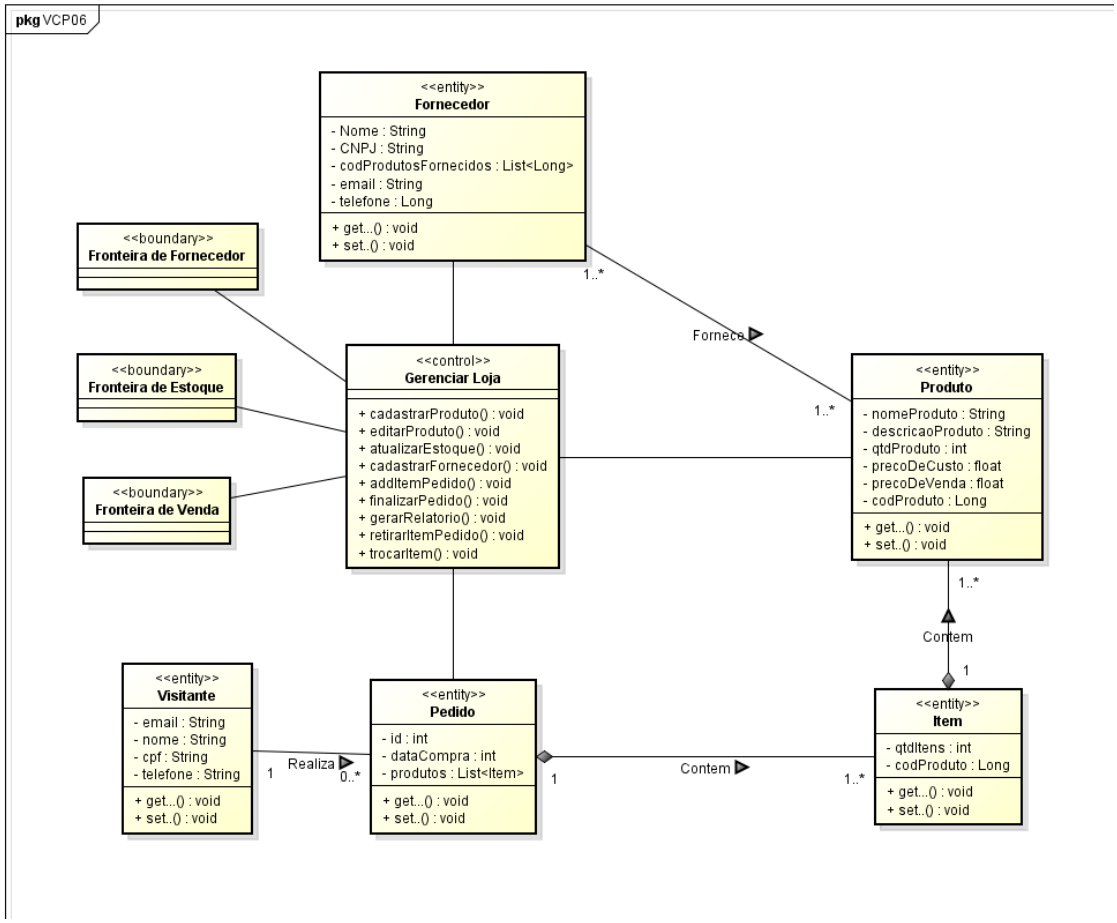
4.5. VCP – CSU05 – Gestão de Evento



Protótipo de tela:

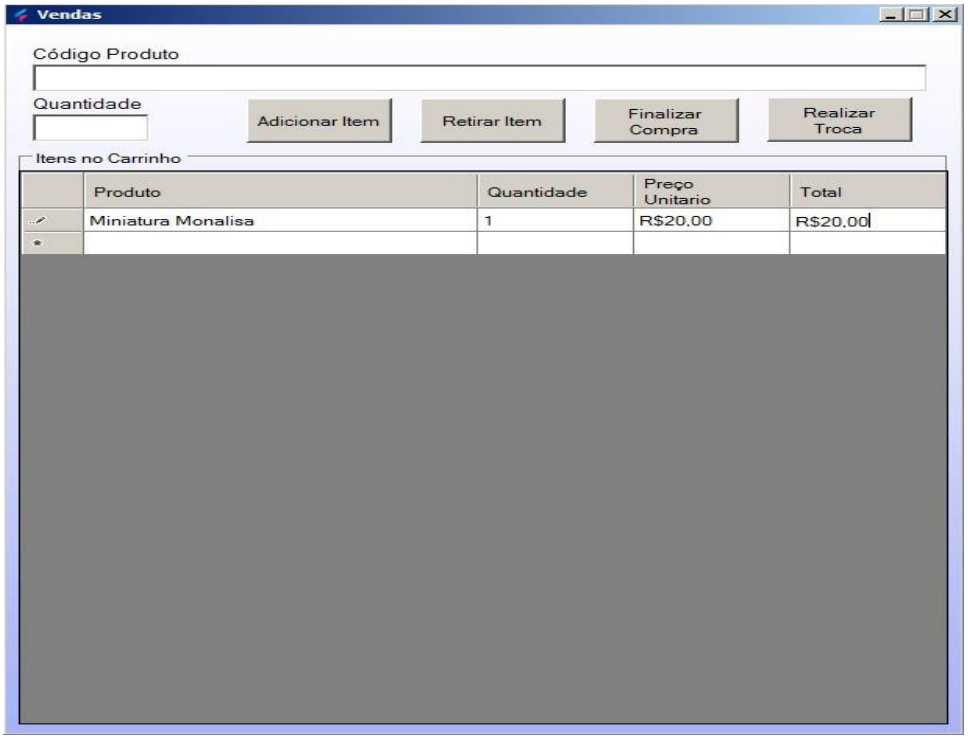


4.6. VCP – CSU06 – Gestão de Venda de Souvenir

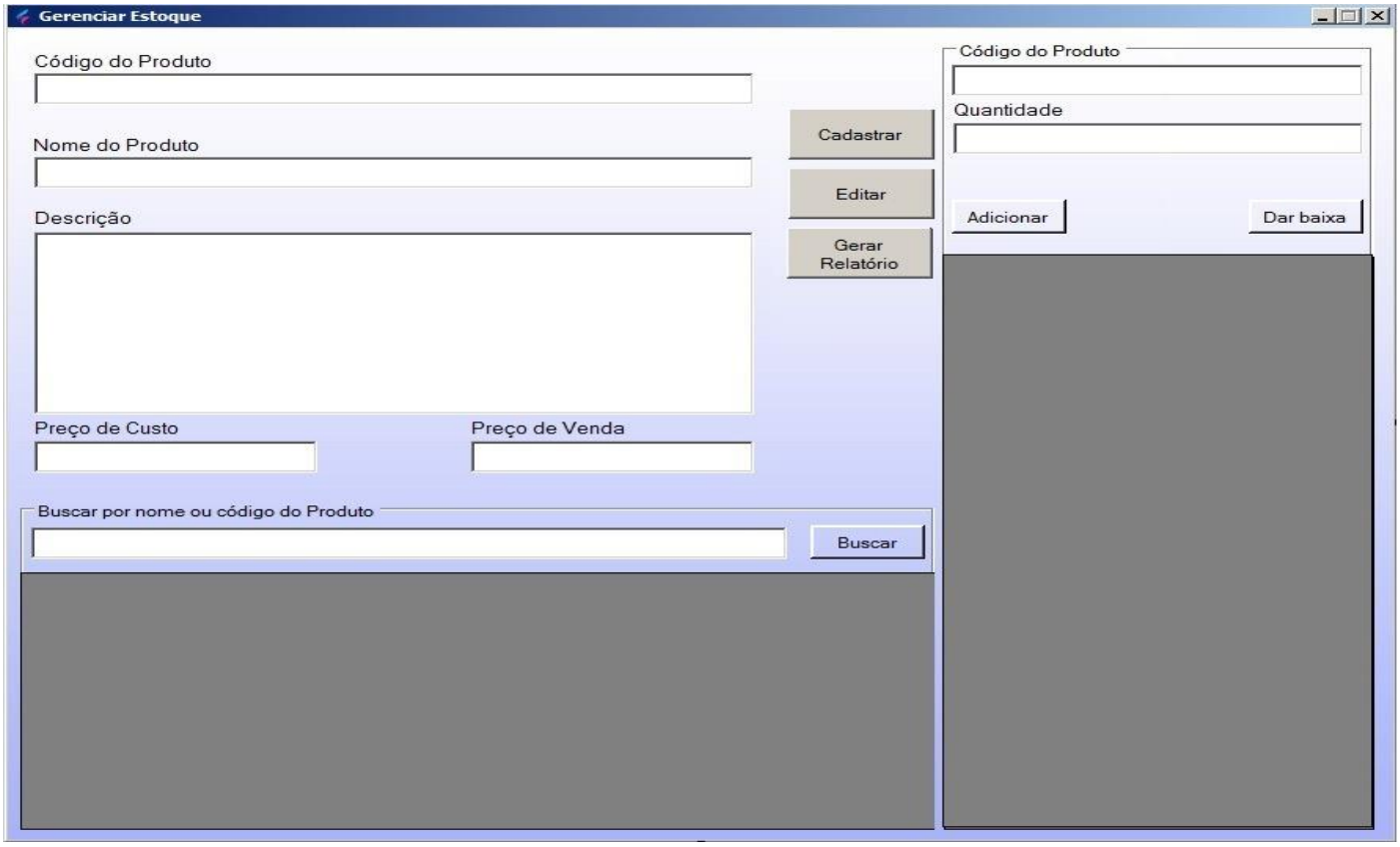


Protótipo das telas:

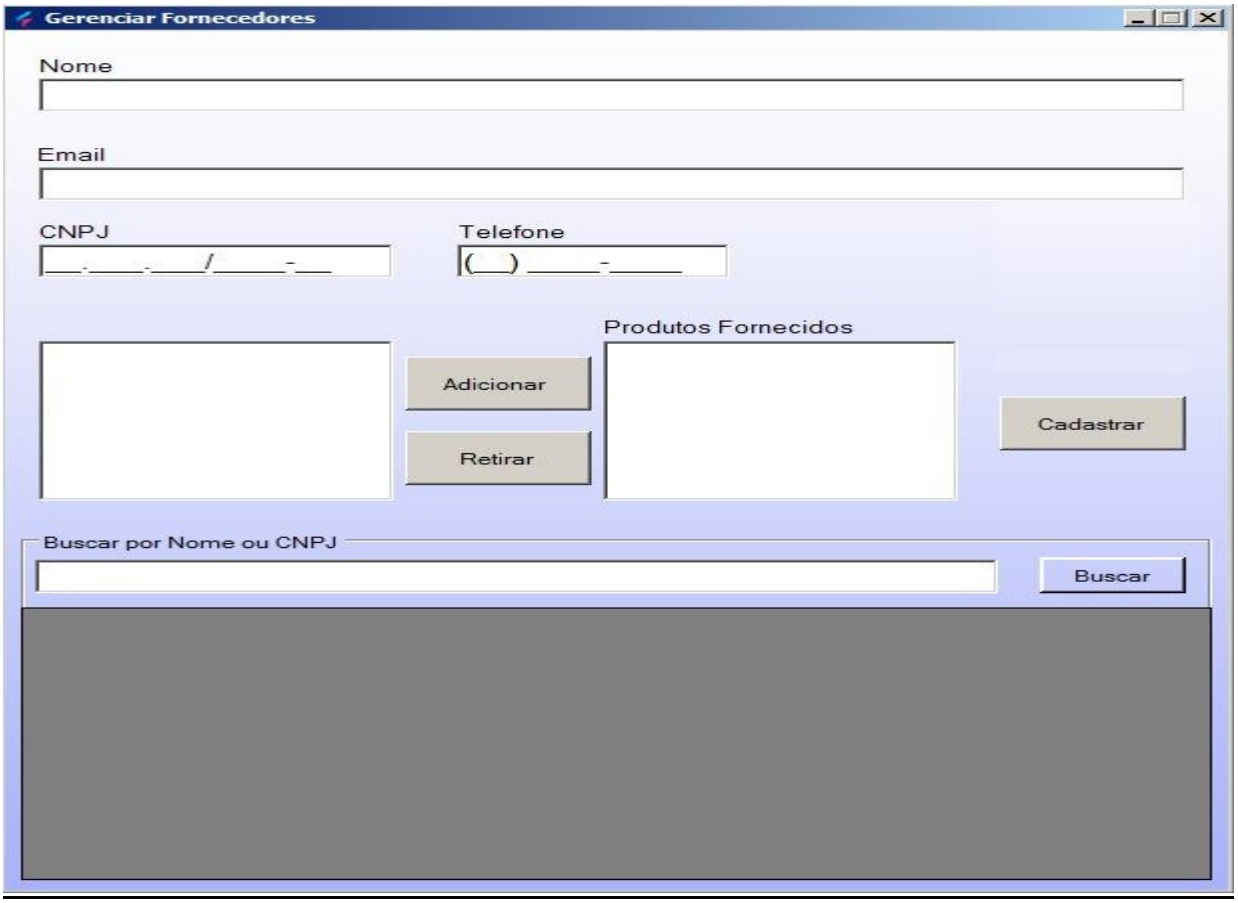
1-Tela de vendas:



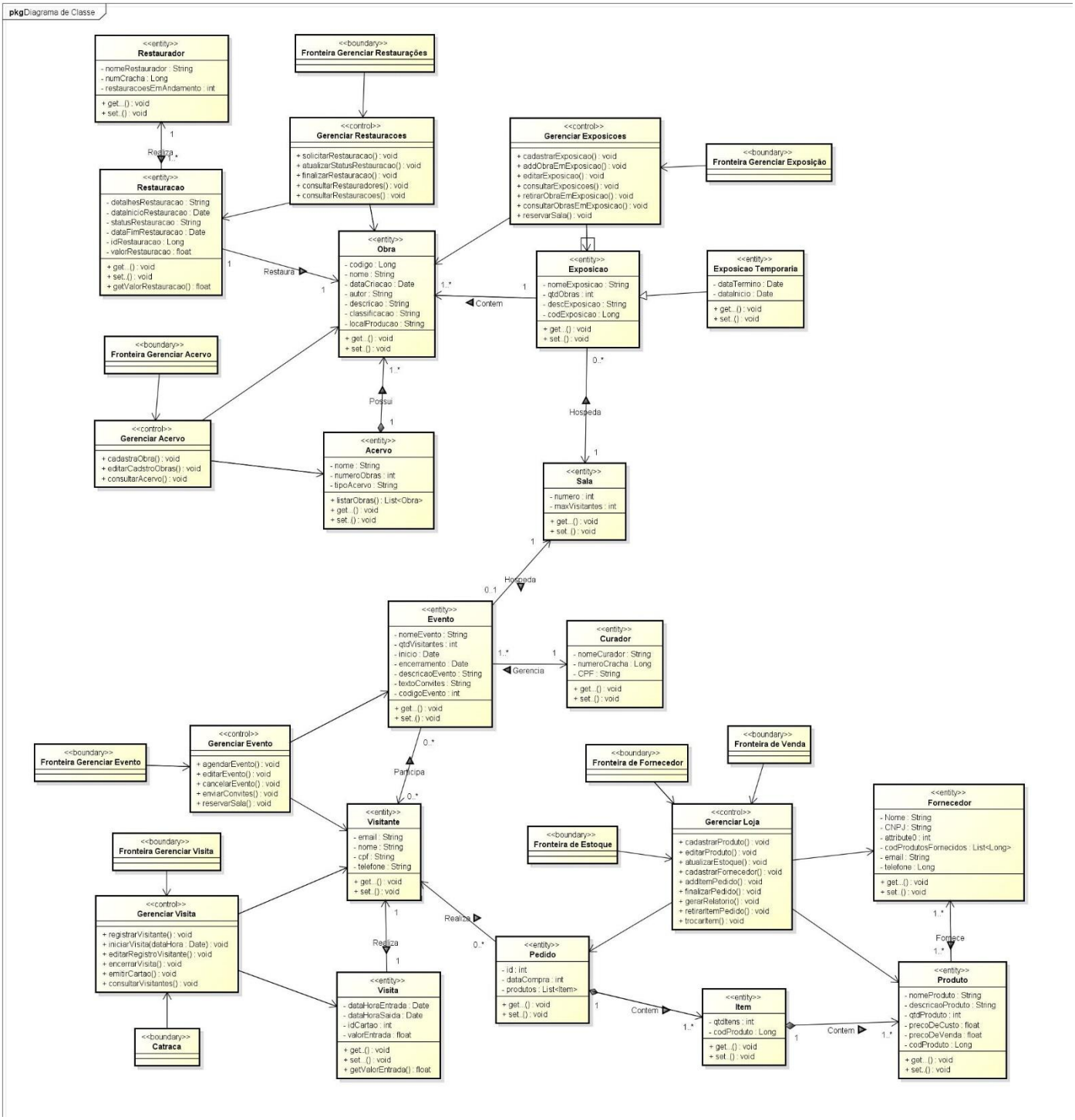
2-Tela de gerenciar estoque:



3-Tela de gerenciar fornecedor:



5. DIAGRAMA DE CLASSES DE PROJETO



6. EXERCÍCIOS TEÓRICOS E DE MODIFICAÇÃO

6.1. Exercício 15

Qual é a classe de entidade mais coesa e a menos coesa do diagrama de classes de projeto? Justifique a tua resposta.

A mesmo coesa é a classe Acervo, pois além de manter informações específicas sobre o acervo, ela também lista as obras, para maior coesão nas entidades, a listagem deveria ser feita no controller. A mais coesa é a classe Produto, pois ela trata das informações de forma que possui certa independência das outras classes, mas claro, ela depende de outras para o negócio de vendas fluir.

6.2. Exercício 16

Qual é a classe de entidade mais acoplada e a menos acoplada do diagrama de classes de projeto? Justifique a tua resposta.

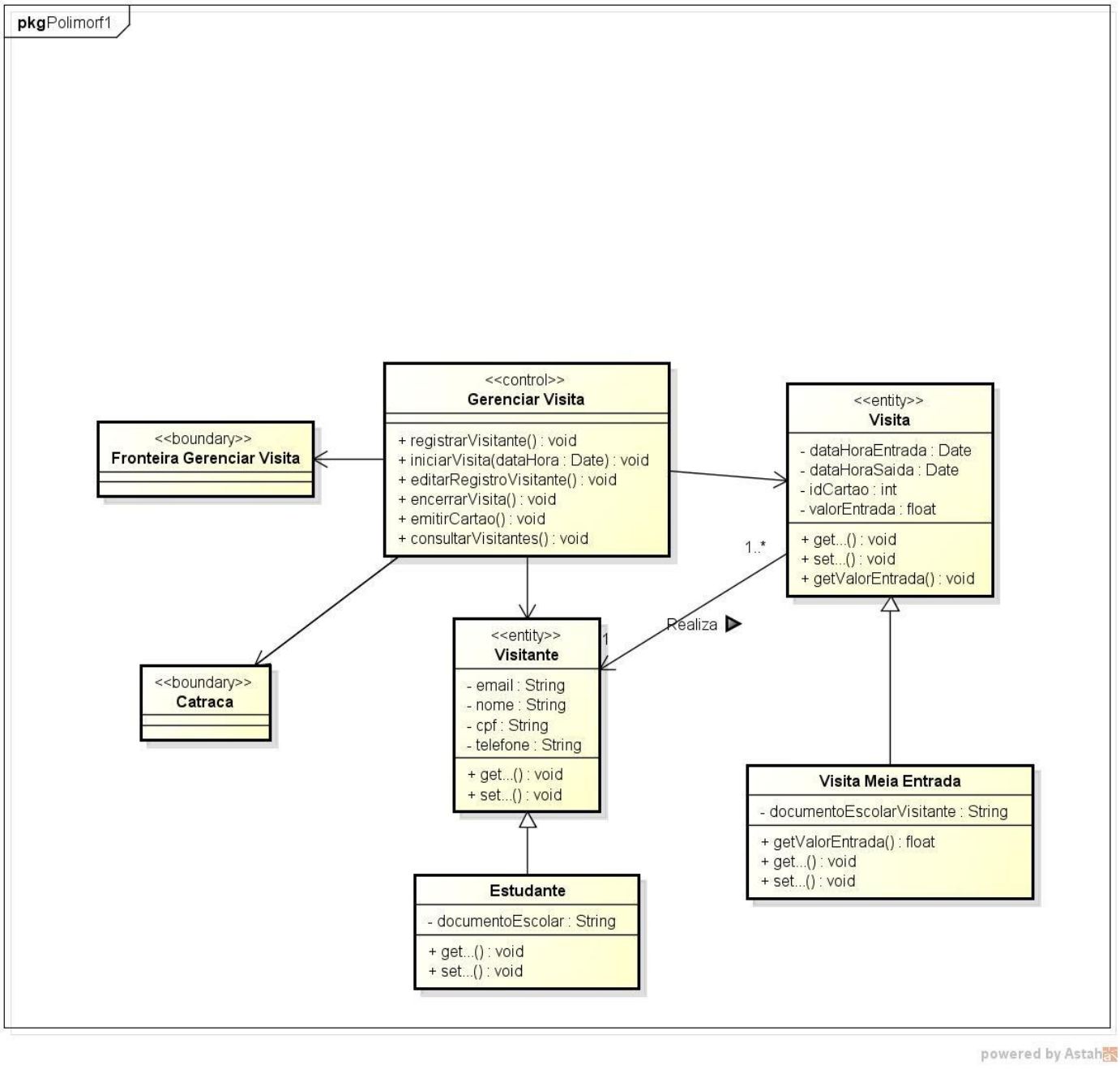
A classe de entidade mais acoplada é a classe Obra, pois essa classe possui relacionamento com a maior quantidade de classes no diagrama (Restauração, Exposição, Acervo, Gerenciar Acervo, Gerenciar Exposições e Gerenciar Restaurações). Já a classe menos acoplada é a classe Curador, pois a mesma possui relacionamento apenas com a classe Evento.

6.3. Exercício 17

Modele duas relações de gen/espec e ative o princípio de polimorfismo universal de inclusão em cada uma delas. Justifique a razão de existência de cada gen/espec e das operações polimórficas. As relações de gen/espec violam o Princípio de Liskov? Justifique a tua resposta.

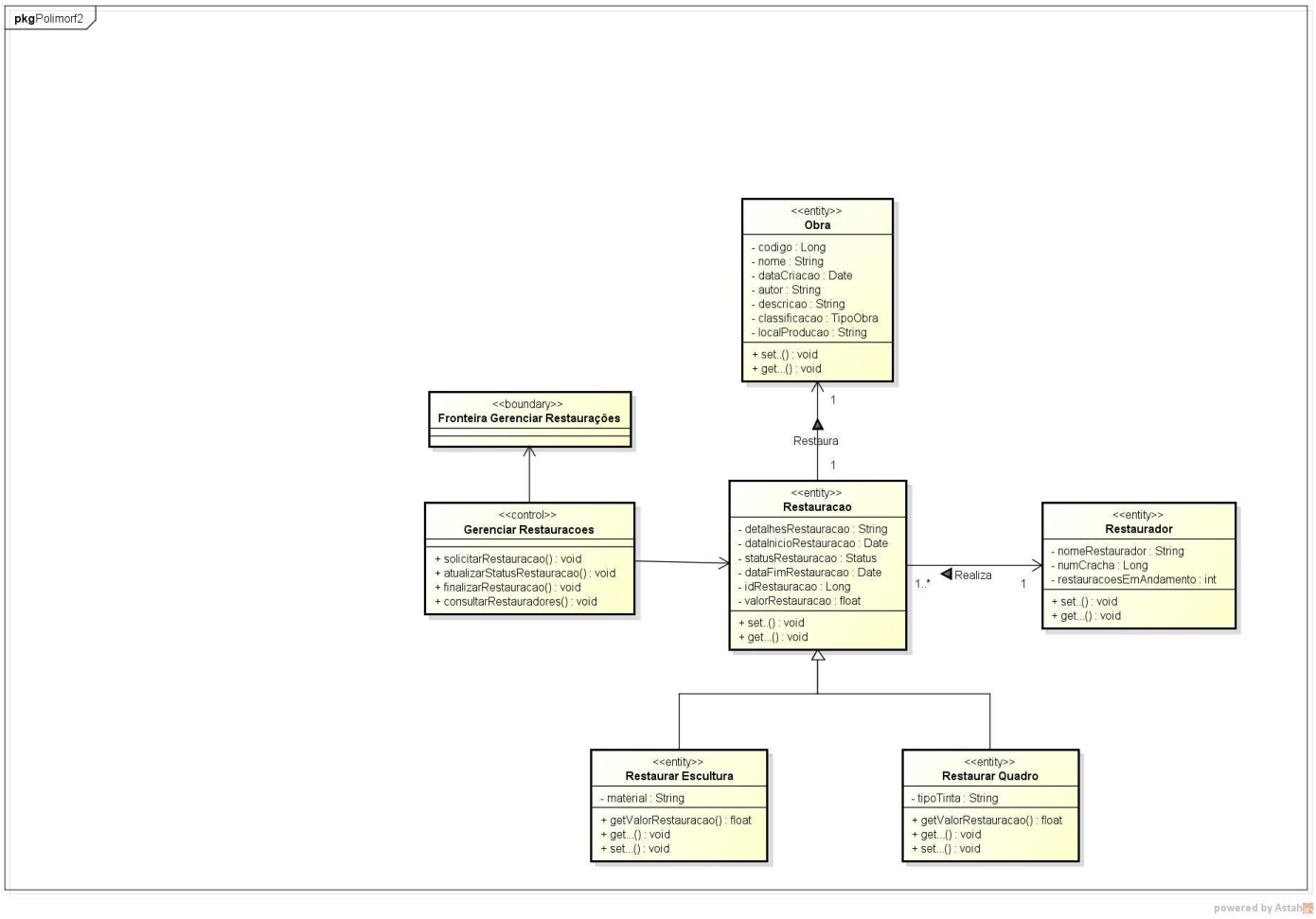
1-Gens/Spec Visita e Visita Meia Entrada: a gen/espec existe devido ao fato de estudantes poderem pagar meia entrada em serviços culturais bastando apenas apresentar o documento escolar que comprove que são estudantes, a operação polimórfica se dá através do método getValorEntrada(), onde na classe mãe retorna o valor do ingresso e na filha retorna metade do valor do ingresso. Essa gen/espec não viola o princípio de Liskov pois uma visita com meia entrada é uma visita. Também foi adicionado gen/espec em visitante e estudante, pois um estudante, além

das informações de visitante, possui também o documento escolar que deve ser adicionado para gerar uma meia entrada. Também não viola o princípio de Liskov pois um estudante é um tipo de visitante.



2-Gen/espec Restauração: a gen/espec existe devido aos tipos específicos de restauração que podem existir, como, no caso, restauração de estatua e quadro. Na gen/espec da restauração de estatuas, a restauração requer o material do qual a estátua é feita e a operação polimórfica é feita no método getValorRestauracao() pois é cobrada uma taxa de 20% a mais para restaurar uma estátua. Na gen/espec da restauração de quadros, a restauração requer o tipo de tinta que foi

utilizado para pintar o quadro a ser restaurado e a operação polimórfica é feita no método getValorRestauracao() pois é cobrada uma taxa de 10% a mais para restaurar um quadro.



6.4. Exercício 18

Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as relações de gen/espec e as operações polimórficas.

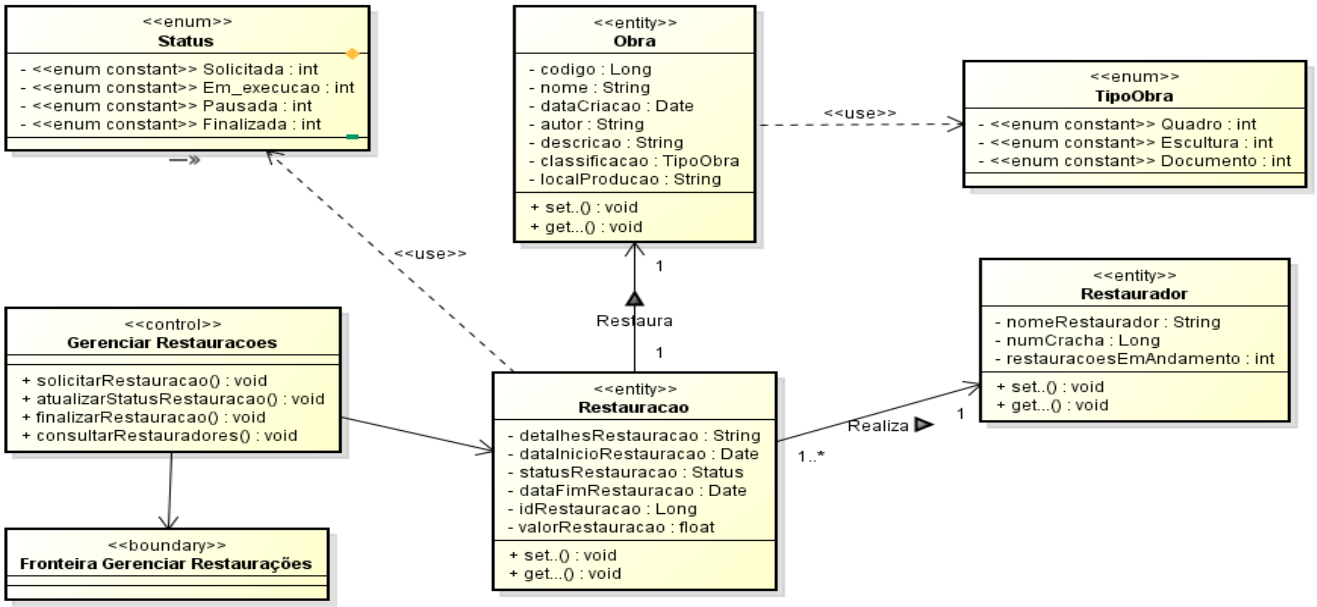
```
public class Visita
{
	public virtual float valorEntrada { get; set; }
}

public class VisitaMeiaEntrada : Visita
{
	public override float valorEntrada
	{
		get { return base.valorEntrada * 0.5f; }
		set { base.valorEntrada = value; }
	}
}

public class Restauracao
{
	public virtual float valorRestauracao { get; set; }
}

public class RestaurarEstatua : Restauracao
{
	public override float valorRestauracao
	{
		get { return base.valorRestauracao * 1.20f; }
		set { base.valorRestauracao = value; }
	}
}

public class RestaurarQuadro : Restauracao
{
	public override float valorRestauracao
	{
		get { return base.valorRestauracao * 1.10f; }
		set { base.valorRestauracao = value; }
	}
}
```

6.5. Exercício 19

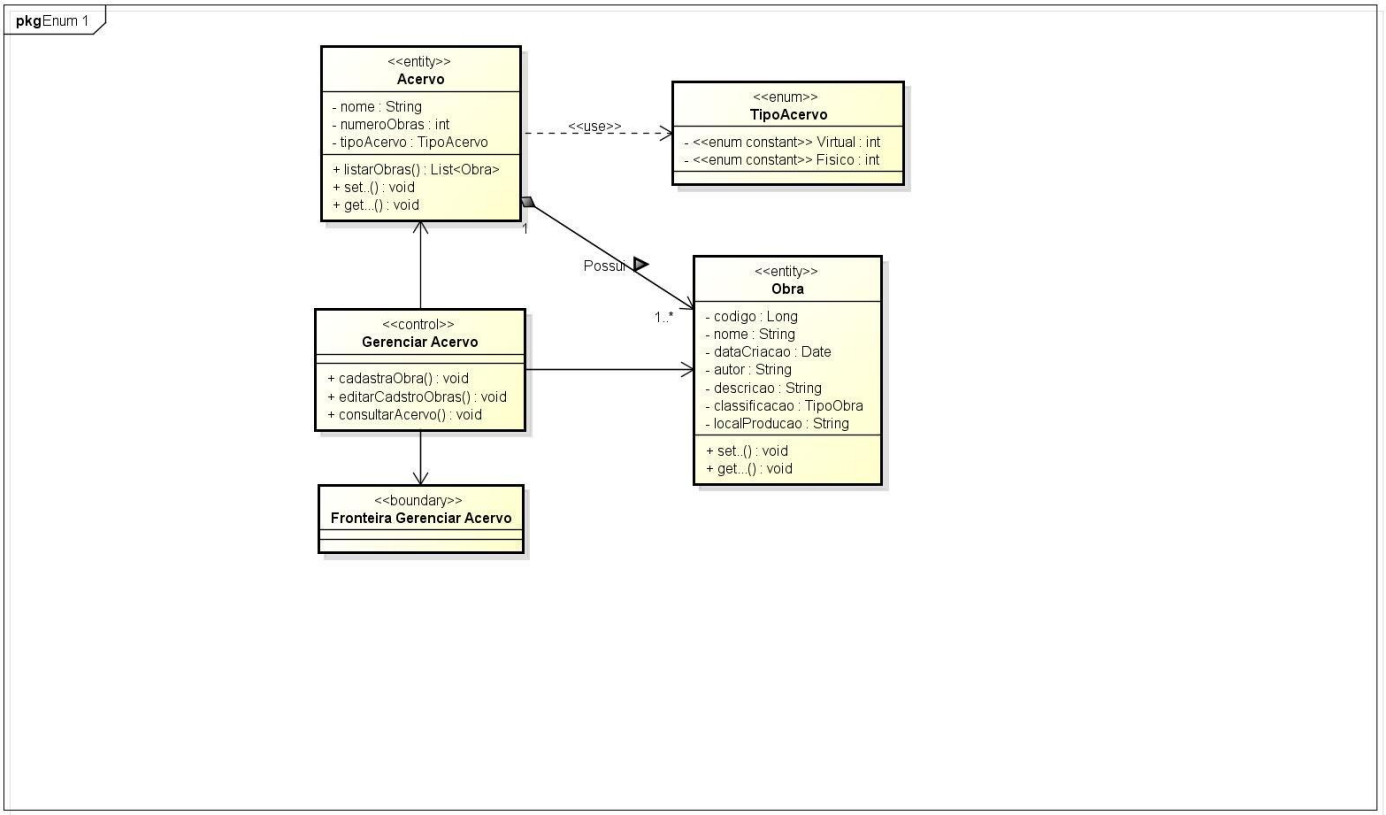
Modele três classes enumeradas e utilize as mesmas como tipos de atributos. Justifique a existência de cada uma das classes enumeradas modeladas.

Enumerada TipoObra: essa classe enumerada existe devido a necessidade de enumerar os diferentes tipos de obras que existem no acervo do museu, no caso o museu pode ter tanto Quadros quanto esculturas e documentos.

Enumerada StatusRestauração: essa classe enumerada existe para haver um melhor controle sobre o status da restauração, com status já enumerados, para que os usuários do sistema não precisem se prender a Strings padrão para identificar o status.



Enumerada TipoAcervo: essa classe enumerada existe para identificar e enumerar qual o tipo de acervo será utilizado, no caso, o acervo Físico contém as estatuas e quadros e o acervo virtual contém os documentos.


pkgEnum 1
<<entity>>
Acervo
- nome : String
- numeroObras : int
- tipoAcervo : TipoAcervo
+ listarObras() : List<Obra>
+ set..() : void
+ get...() : void
<<use>>
<<enum>>
TipoAcervo
- <<enum constant>> Virtual : int
- <<enum constant>> Fisico : int
1
Possui
1..*
<<entity>>
Obra
- codigo : Long
- nome : String
- dataCriacao : Date
- autor : String
- descricao : String
- classificacao : TipoObra
- localProducao : String
+ set..() : void
+ get...() : void
<<control>>
Gerenciar Acervo
+ cadastraObra() : void
+ editarCadstroObras() : void
+ consultarAcervo() : void
<<boundary>>
Fronteira Gerenciar Acervo
powered by Astah

6.6. Exercício 20

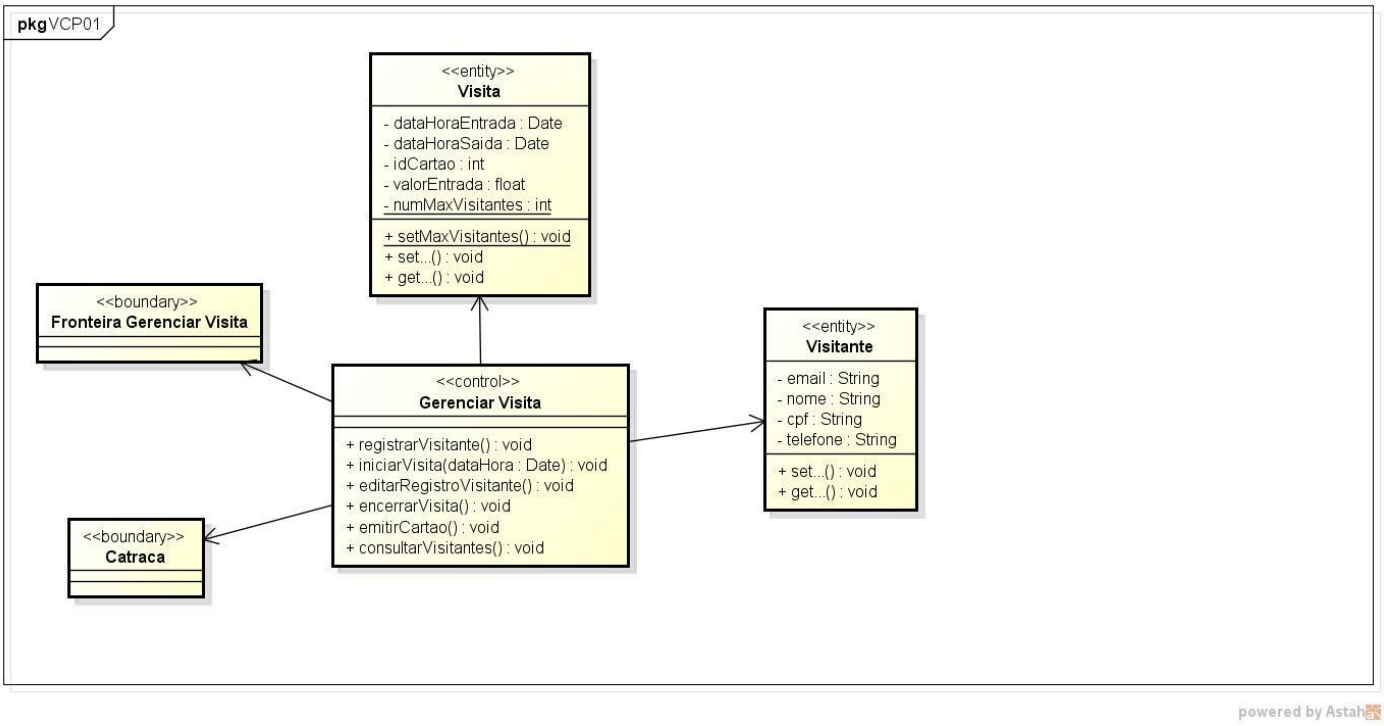
Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as três classes enumeradas.

```
public enum StatusRestauracao
{
	SOLICITADA = 0,
	EM_EXECUCAO = 1,
	PAUSADA = 2,
	FINALIZADA = 3
}

public enum TipoAcervo
{
	VIRTUAL = 0,
	FISICO = 1
}

public enum TipoObra
{
	QUADRO = 0,
	ESCULTURA = 1,
	DOCUMENTO = 2
}

public class Acervo
{
	public TipoAcervo tipoAcervo { get; set; }
}

public class Obra
{
	public TipoObra tipoObra { get; set; }
}

public class Restauracao
{
	public StatusRestauracao statusRestauracao { get; set; }
}
```

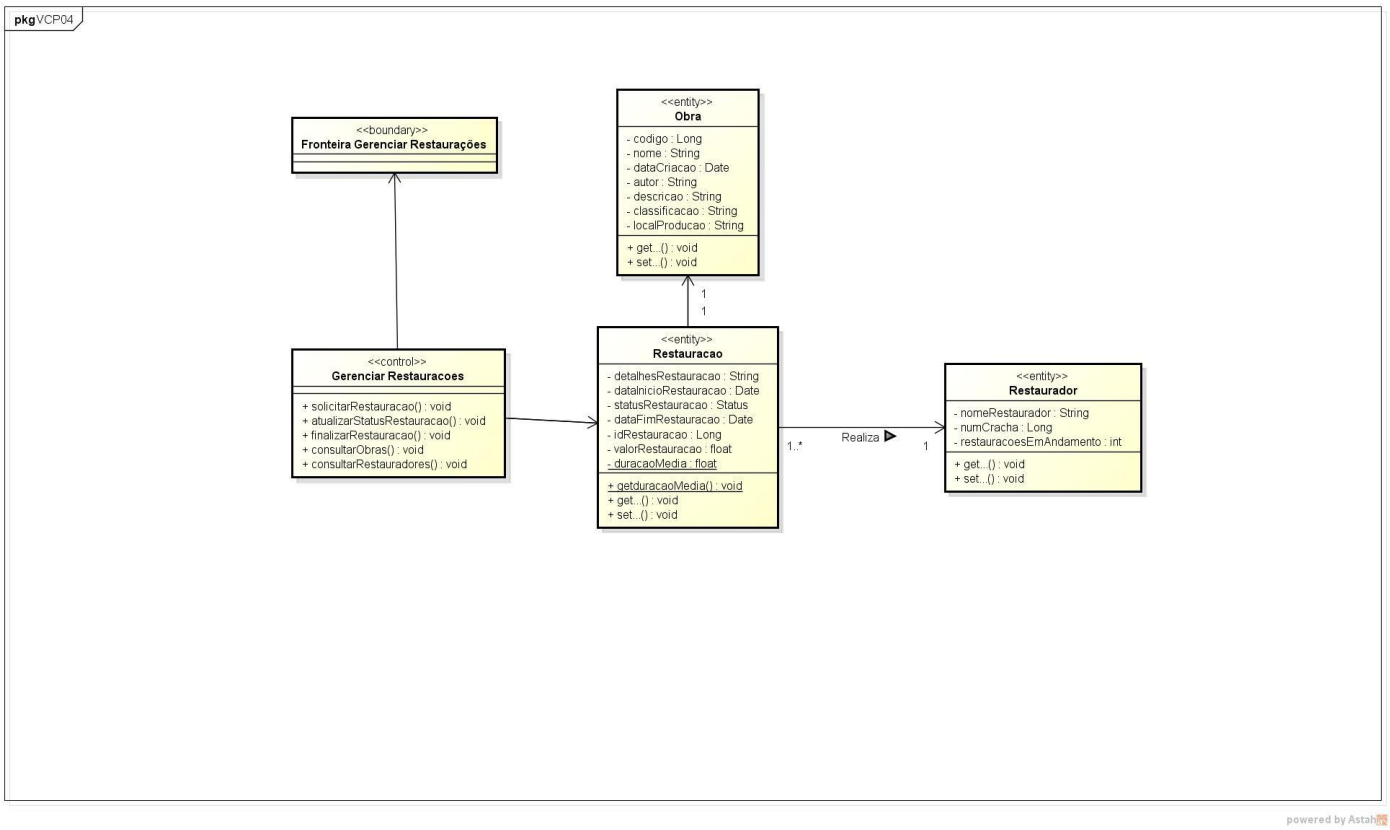
6.7. Exercício 21

Modele seis membros estáticos, sendo três atributos e três métodos. Justifique a criação de existência de cada um dos membros estáticos modelados.

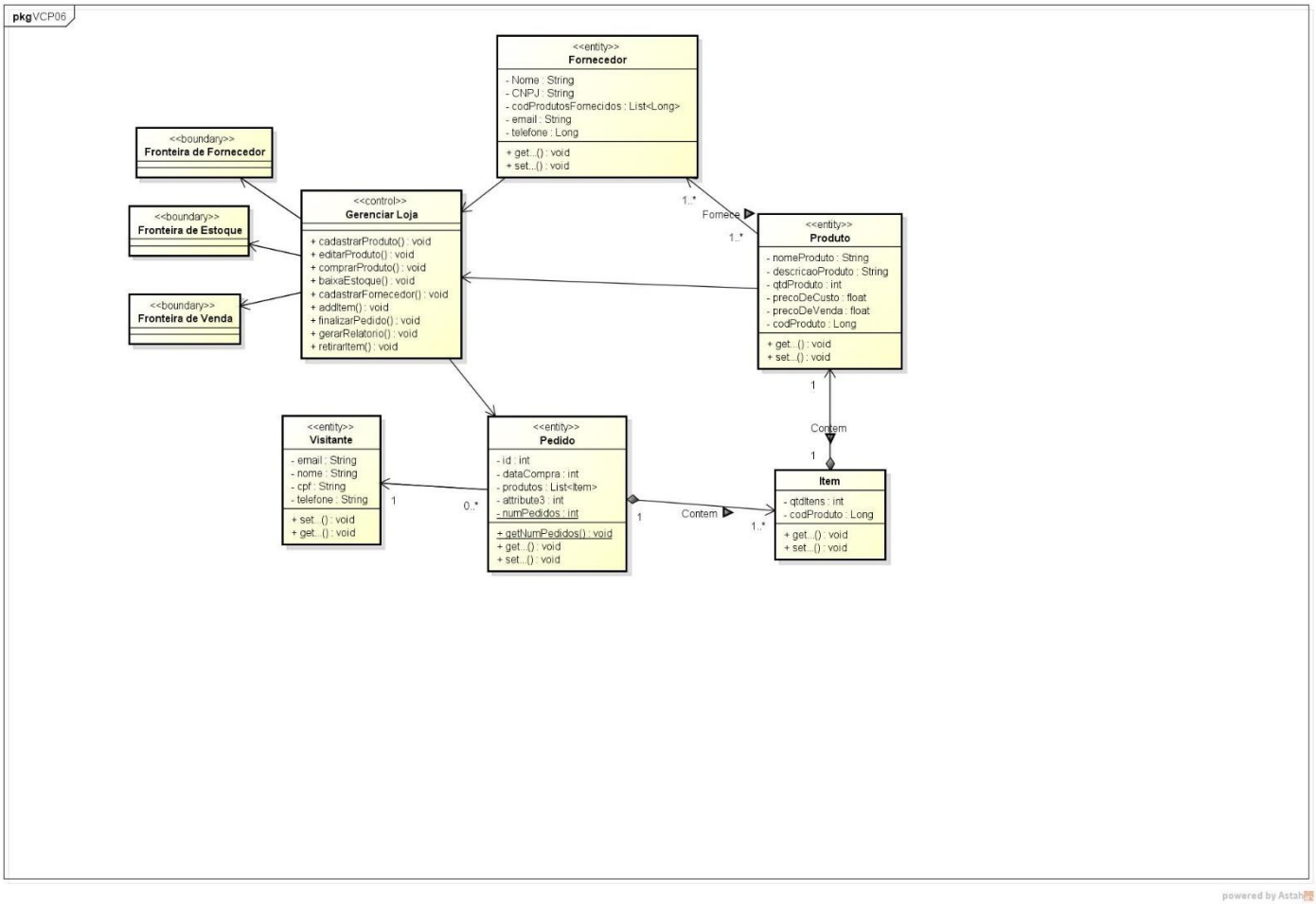
1. Na Entidade Visita o atributo estático numMaxVisitantes, com esse atributo é possível saber qual é a capacidade máxima do museu em número de visitantes. Com o método estático setMaxVisitantes é possível determinar esse valor conforme o tamanho do museu.



2. Na Entidade Restauração o atributo duracaoMedia, esse atributo é determinado pelo cálculo do tempo médio gasto com as restaurações realizadas, para que seja possível saber uma estimativa do tempo que a restauração pode durar. O seu respectivo método, getduracaoMedia retorna esse valor.



3 Na Entidade Pedido o atributo numPedidos é determinado pela contagem de pedidos já realizados, esse valor é importante na geração do relatório de vendas. O atributo possui seu respectivo método getNumPedidos que retorna essa contagem.



6.8. Exercício 22

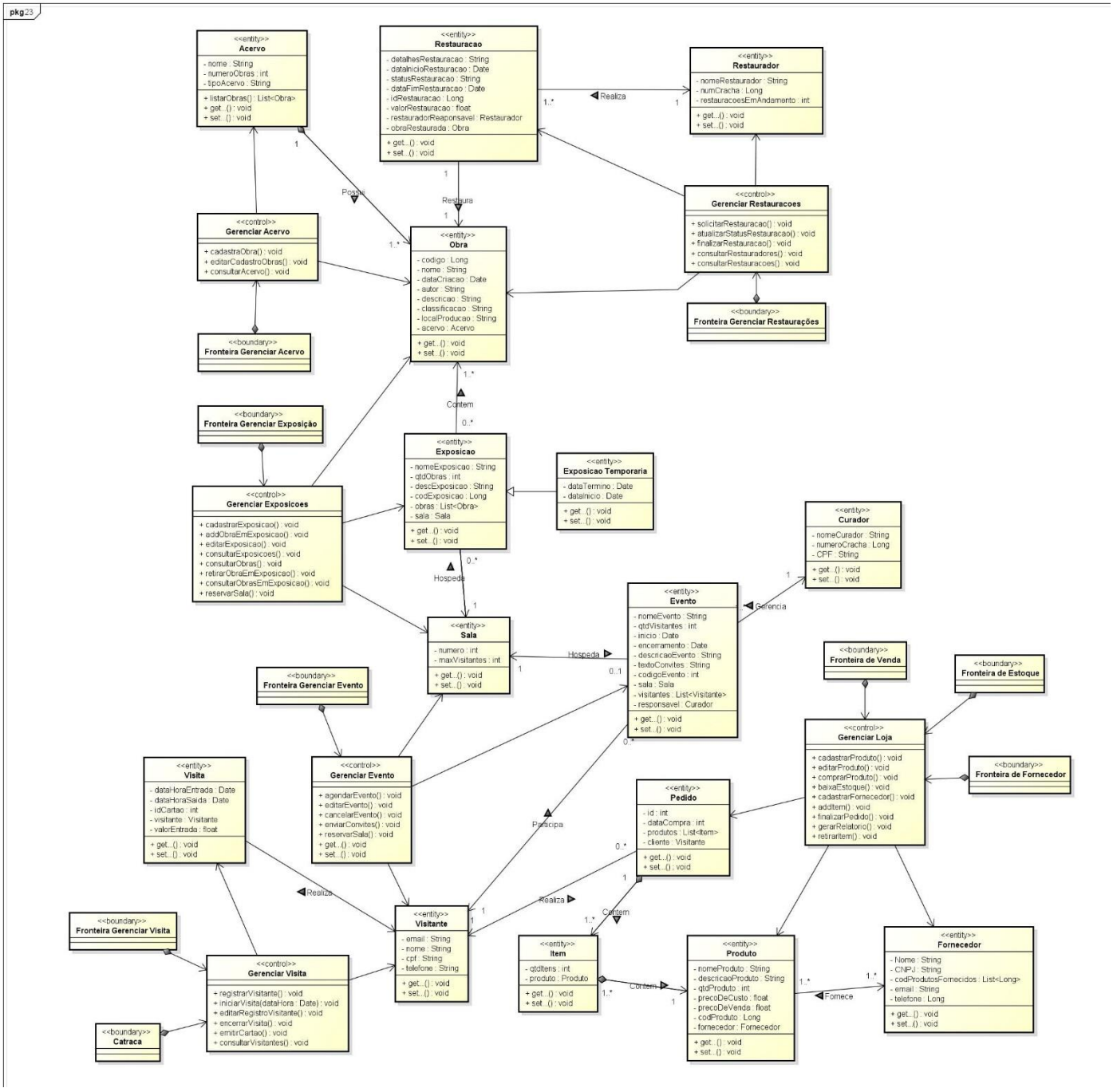
Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar os seis membros estáticos.

```
public class Visita
{
      public static int numMaxVisitantes { get; set; }
}

public class Restauracao
{
      public static float duracaoMedia { get; set; }
}

public class Pedido
{
      public static int numPedidos { get; set; }
}
```

6.9. Exercício 23

Transforme todos os relacionamentos de associação ou agregação entre as classes de entidade e todos os relacionamentos de associação entre as classes de fronteira e controle para dependências estruturais. Explique a vantagem e desvantagem desse tipo de dependência.

A dependência por atributo é uma dependência mais forte e aumenta o desempenho da aplicação porque os objetos que serão necessários já estarão instanciados, porém, expõe toda a classe, o que diminui o encapsulamento e alto nível de acoplamento pois é preciso que a classe faça muitos relacionamentos. (Diagrama na página seguinte)


pkg23
<<entity>>
Acervo
- nome : String
- numeroObras : int
- tipoAcervo : String
+ listarObras() : List<Obra>
+ get...() : void
+ set...() : void
<<entity>>
Restauracao
- detalhesRestauracao : String
- dataInicioRestauracao : Date
- statusRestauracao : String
- dataFimRestauracao : Date
- idRestauracao : Long
- valorRestauracao : float
- restauradorResponsavel : Restaurador
- obraRestaurada : Obra
+ get...() : void
+ set...() : void
<<entity>>
Restaurador
- nomeRestaurador : String
- numCracha : Long
- restauracoesEmAndamento : int
+ get...() : void
+ set...() : void
Realiza
Possui
Restaura
<<control>>
Gerenciar Acervo
+ cadastraObra() : void
+ editarCadastroObras() : void
+ consultarAcervo() : void
<<entity>>
Obra
- codigo : Long
- nome : String
- dataCriacao : Date
- autor : String
- descricao : String
- classificacao : String
- localProducao : String
- acervo : Acervo
+ get...() : void
+ set...() : void
<<control>>
Gerenciar Restauracoes
+ solicitarRestauracao() : void
+ atualizarStatusRestauracao() : void
+ finalizarRestauracao() : void
+ consultarRestauradores() : void
+ consultarRestauracoes() : void
<<boundary>>
Fronteira Gerenciar Acervo
<<boundary>>
Fronteira Gerenciar Restaurações
Contem
<<boundary>>
Fronteira Gerenciar Exposição
<<entity>>
Exposicao
- nomeExposicao : String
- qtdObras : int
- descExposicao : String
- codExposicao : Long
- obras : List<Obra>
- sala : Sala
+ get...() : void
+ set...() : void
<<entity>>
Exposicao Temporaria
- dataTermino : Date
- dataInicio : Date
+ get...() : void
+ set...() : void
<<control>>
Gerenciar Exposicoes
+ cadastrarExposicao() : void
+ addObraEmExposicao() : void
+ editarExposicao() : void
+ consultarExposicoes() : void
+ consultarObras() : void
+ retirarObraEmExposicao() : void
+ consultarObrasEmExposicao() : void
+ reservarSala() : void
Hospeda
<<entity>>
Curador
- nomeCurador : String
- numeroCracha : Long
- CPF : String
+ get...() : void
+ set...() : void
Gerencia
<<entity>>
Sala
- numero : int
- maxVisitantes : int
+ get...() : void
+ set...() : void
Hospeda
<<entity>>
Evento
- nomeEvento : String
- qtdVisitantes : int
- inicio : Date
- encerramento : Date
- descricaoEvento : String
- textoConvites : String
- codigoEvento : int
- sala : Sala
- visitantes : List<Visitante>
- responsavel : Curador
+ get...() : void
+ set...() : void
<<boundary>>
Fronteira de Venda
<<boundary>>
Fronteira de Estoque
<<boundary>>
Fronteira Gerenciar Evento
<<control>>
Gerenciar Loja
+ cadastrarProduto() : void
+ editarProduto() : void
+ comprarProduto() : void
+ baixaEstoque() : void
+ cadastrarFornecedor() : void
+ addItem() : void
+ finalizarPedido() : void
+ gerarRelatorio() : void
+ retirarItem() : void
<<boundary>>
Fronteira de Fornecedor
<<entity>>
Visita
- dataHoraEntrada : Date
- dataHoraSaida : Date
- idCartao : int
- visitante : Visitante
- valorEntrada : float
+ get...() : void
+ set...() : void
<<control>>
Gerenciar Evento
+ agendarEvento() : void
+ editarEvento() : void
+ cancelarEvento() : void
+ enviarConvites() : void
+ reservarSala() : void
+ get...() : void
+ set...() : void
<<entity>>
Pedido
- id : int
- dataCompra : int
- produtos : List<Item>
- cliente : Visitante
+ get...() : void
+ set...() : void
Participa
Realiza
Realiza
Contem
<<boundary>>
Fronteira Gerenciar Visita
<<entity>>
Visitante
- email : String
- nome : String
- cpf : String
- telefone : String
+ get...() : void
+ set...() : void
<<entity>>
Item
- qtdItens : int
- produto : Produto
+ get...() : void
+ set...() : void
Contem
<<entity>>
Produto
- nomeProduto : String
- descricaoProduto : String
- qtdProduto : int
- precoDeCusto : float
- precoDeVenda : float
- codProduto : Long
- fornecedor : Fornecedor
+ get...() : void
+ set...() : void
Fornece
<<entity>>
Fornecedor
- Nome : String
- CNPJ : String
- codProdutosFornecidos : List<Long>
- email : String
- telefone : Long
+ get...() : void
+ set...() : void
<<control>>
Gerenciar Visita
+ registrarVisitante() : void
+ iniciarVisita(dataHora : Date) : void
+ editarRegistroVisitante() : void
+ encerrarVisita() : void
+ emitirCartao() : void
+ consultarVisitantes() : void
<<boundary>>
Catraca

6.10. Exercício 24

Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências estruturais.

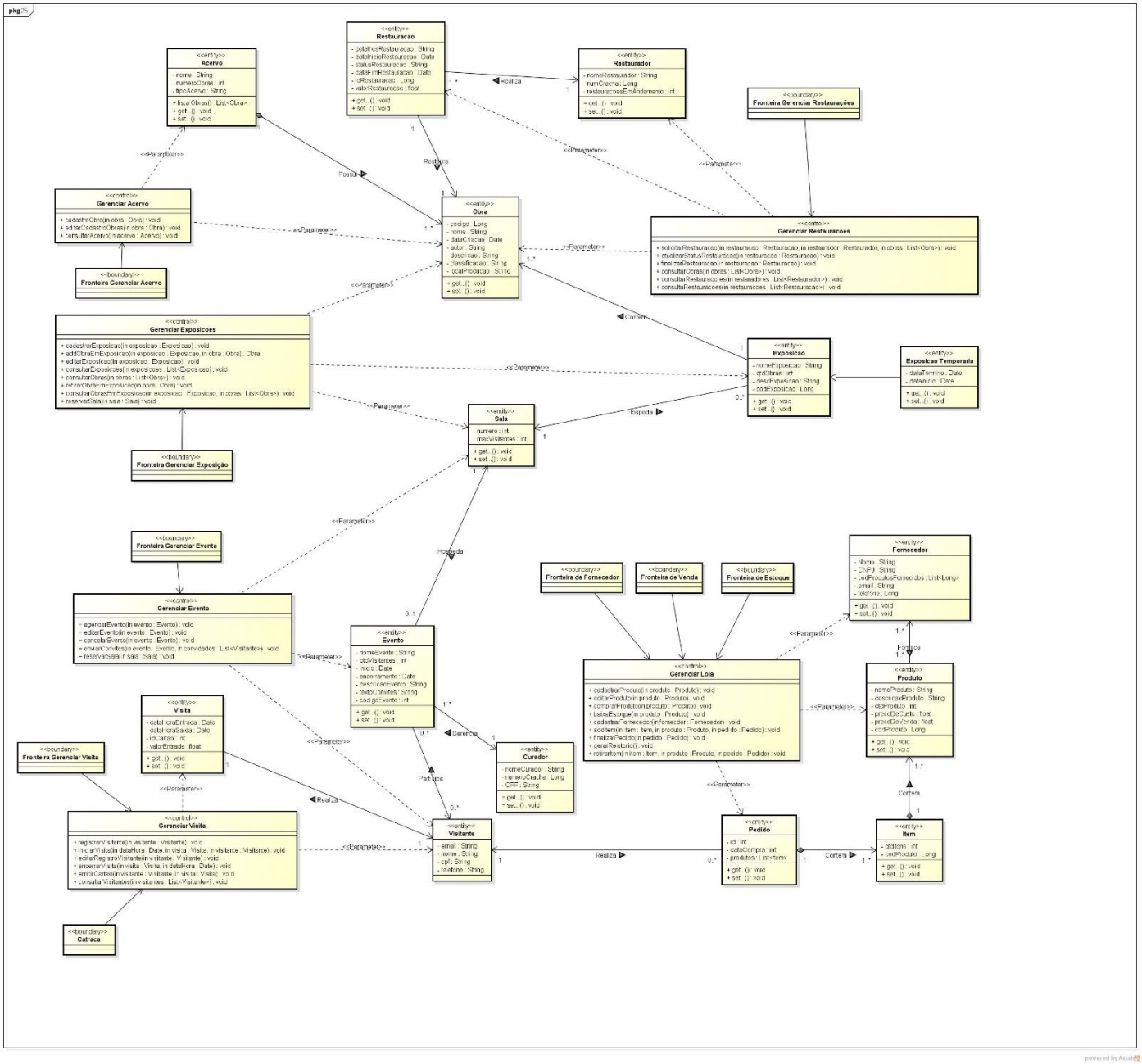
```
// Fronteiras
public class FronteiraDeEstoque
{
	public GerenciarLoja gerenciarLoja { get; set; }
}

public class FronteiraDeVenda
{
	public GerenciarLoja gerenciarLoja { get; set; }
}

public class FronteiraGerenciarAcervo
{
	public GerenciarAcervo gerenciarAcervo { get; set; }
}

public class FronteiraGerenciarEvento
{
	public GerenciarEvento gerenciarEvento { get; set; }
}

public class FronteiraGerenciarExposicao
{
	public GerenciarExposicoes gerenciarExposicoes { get; set; }
}

public class FronteiraGerenciarRestauracoes
{
	public GerenciarRestauracoes gerenciarRestauracoes { get; set; }
}

public class FronteiraGerenciarVisita
{
	public GerenciarVisita gerenciarVisita { get; set; }
}

public class Catraca
{
	public GerenciarVisita gerenciarVisita { get; set; }
}

// Entidades
public class Acervo
{
	public IList<Obra> obras { get; set; }
}

public class Curador {}
```

```
public class Evento
{
      public Curador responsavel { get; set; }
      public IList<Visitante> visitantes { get; set; }
}

public class Exposicao
{
      public Sala sala { get; set; }
}

public class ExposicaoTemporaria : Exposicao
{

}

public class Fornecedor
{
      public IList<Produto> produtos { get; set; }
}

public class Item
{
      public Produto produto { get; set; }
      public Pedido pedido { get; set; }
}

public class Obra
{
      public Acervo acervo { get; set; }
}

public class Pedido
{
      public Visitante cliente { get; set; }
      public IList<Item> produtos { get; set; }
}

public class Produto
{
      public Fornecedor fornecedor { get; set; }
}

public class Restauracao
{
      public Restaurador restauradorResponsavel { get; set; }
      public Obra obraRestaurada { get; set; }
}

public class Visita
{
      public Visitante visitante { get; set; }
}
```

6.11. Exercício 25

Transforme todos os relacionamentos de associação entre as classes de controle e entidade para dependências não estruturais por parâmetro. Explique a vantagem e desvantagem desse tipo de dependência.

Uma vantagem da dependência por parâmetro é o aumento do encapsulamento porque a dependência por atributo não acontece, ou seja, o controle de acesso aos atributos e métodos ficam mais restritos, porém cai o desempenho da aplicação porque um objeto recebe outro como parâmetro em um método e diminui o acoplamento pois a classe possui poucos relacionamentos com as demais classes.



6.12. Exercício 26

Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências não estruturais por parâmetro.

```
public class GerenciarAcervo
{
	public void cadastraObra(Obra obra)
	{
		// [[Implementação aqui]]
	}

	public void editarCadastroObra(Obra obra)
	{
		// [[Implementação aqui]]
	}

	public IList<Obra> consultarAcervo(Acervo acervo)
	{
		// [[Implementação aqui]]
	}
}

public class GerenciarEvento
{
	public void agendarEvento(Evento evento)
	{
		// [[Implementação aqui]]
	}

	public void editarEvento(Evento evento)
	{
		// [[Implementação aqui]]
	}

	public void cancelarEvento(Evento evento)
	{
		// [[Implementação aqui]]
	}

	public void enviarConvites(Evento evento, List<Visitante> convidados)
	{
		// [[Implementação aqui]]
	}

	public void reservarSala(Sala sala)
	{
		// [[Implementação aqui]]
	}
}

public class GerenciarExposicoes
{
	public void cadastrarExposicao(Exposicao exposicao)
	{
```

```
            // [[Implementação aqui]]
      }

      public void addObraEmExposicao(Exposicao exposicao, Obra obra)
      {
            // [[Implementação aqui]]
      }

      public void editarExposicao(Exposicao exposicao)
      {
            // [[Implementação aqui]]
      }

      public IList<Exposicao> consultarExposicoes()
      {
            // [[Implementação aqui]]
      }

      public void retirarObraEmExposicao(Obra obra)
      {
            // [[Implementação aqui]]
      }

      public IList<Obra> consultarObrasEmExposicao(Exposicao exposicao)
      {
            // [[Implementação aqui]]
      }

      public void reservarSala(Sala sala)
      {
            // [[Implementação aqui]]
      }
}

public class GerenciarLoja
{
      public void cadastrarProduto(Produto produto)
      {
            // [[Implementação aqui]]
      }

      public void editarProduto(Produto produto)
      {
            // [[Implementação aqui]]
      }

      public void comprarProduto(Produto produto, int quantidade, float preco)
      {
            // [[Implementação aqui]]
      }

      public void baixaEstoque(Produto produto, int quantidade)
      {
            // [[Implementação aqui]]
      }
```

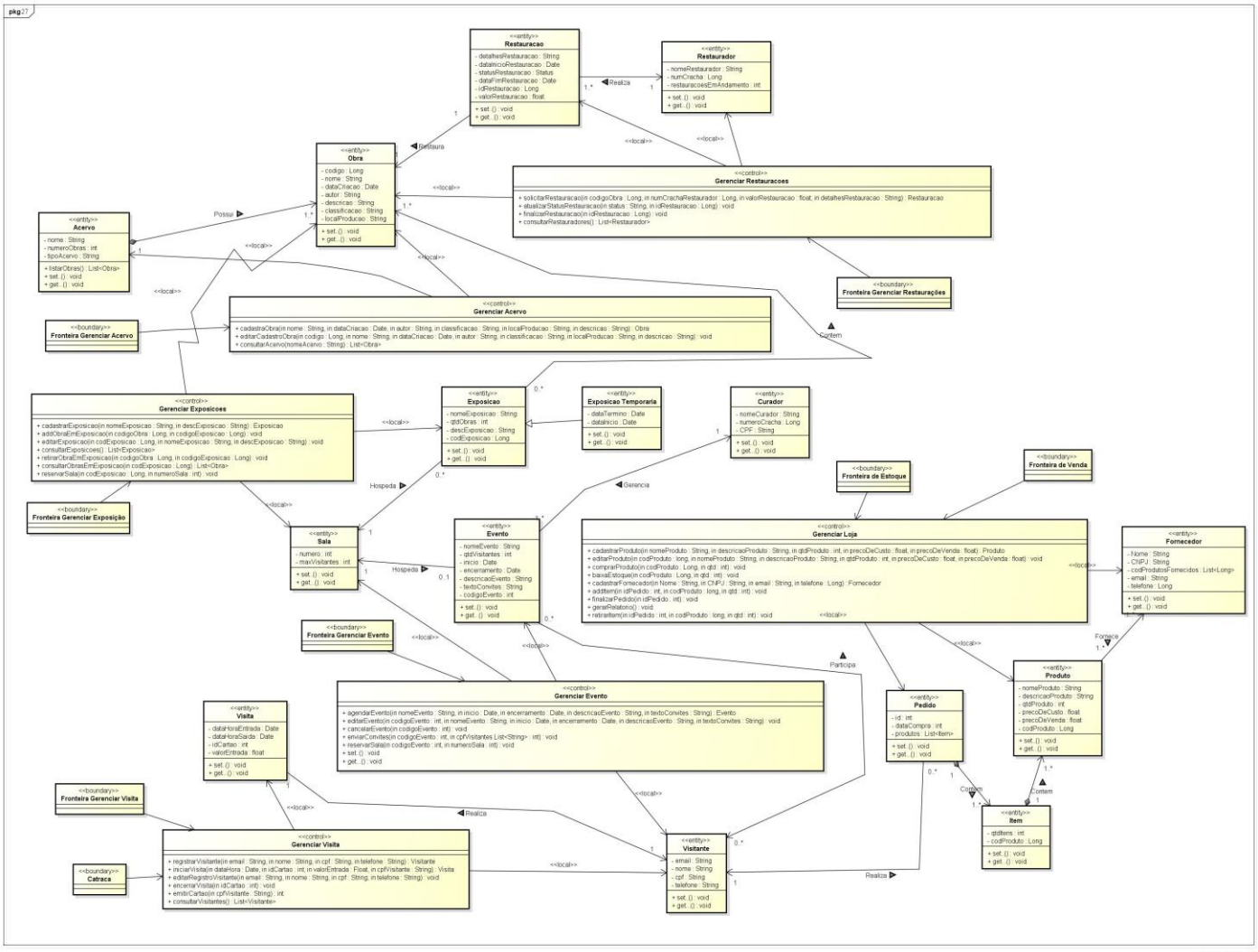
```
	public void cadastrarFornecedor(Fornecedor fornecdor)
	{
		// [[Implementação aqui]]
	}

	public void addItem(Item item, Produto produto, Pedido pedido)
	{
		// [[Implementação aqui]]
	}

	public void finalizarPedido(Pedido pedido)
	{
		// [[Implementação aqui]]
	}

	public void gerarRelatorio()
	{
		// [[Implementação aqui]]
	}

	public void retirarItem(Item item, Produto produto, Pedido pedido)
	{
		// [[Implementação aqui]]
	}
}

public class GerenciarRestauracoes
{
	public void solicitarRestauracao(Restauracao restauracao, Restaurador
restaurador, List<Obra> obras)
	{
		// [[Implementação aqui]]
	}

	public void atualizarStatusRestauracao(Restauracao restauracao)
	{
		// [[Implementação aqui]]
	}

	public void finalizarRestauracao(Restauracao restauracao)
	{
		// [[Implementação aqui]]
	}

	public IList<Restauracao> consultarRestauracoes(Restauracao restauracao)
	{
		// [[Implementação aqui]]
	}
	public IList<Restauracao> consultarRestauradores(Restaurador restaurador)
	{
		// [[Implementação aqui]]
	}


}
```

```
public class GerenciarVisita
{
	public void registrarVisitante(Visitante visitante)
	{
		// [[Implementação aqui]]
	}

	public void iniciarVisita(DateTime dataHora, Visita visita, Visitante visitante)
	{
		// [[Implementação aqui]]
	}

	public void editarRegistroVisitante(Visitante visitante)
	{
		// [[Implementação aqui]]
	}

	public void encerrarVisita(Visita visita, DateTime dataHora)
	{
		// [[Implementação aqui]]
	}

	public void emitirCartao(Visitante visitante, Visita visita)
	{
		// [[Implementação aqui]]
	}

	public IList<Visitante> consultarVisitantes()
	{
		// [[Implementação aqui]]
	}
}
```

6.13. Exercício 27

Transforme todos os relacionamentos de associação entre as classes de controle e entidade para dependências não estruturais por variável local. Explique a vantagem e desvantagem desse tipo de dependência.

- **Vantagem:** Economia de memória, pois a variável só existe em tempo de execução.
- **Desvantagem:** A alteração no método principal pode afetar os métodos dependentes.



6.14. Exercício 28

Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências não estruturais por variável local.

```
public class GerenciarAcervo
{
	public Obra cadastraObra(String nome, DateTime dataCriacao, String autor, String
classificacao, String localProducao, String descricao)
	{
		Obra obra = new Obra();
		obra.nome = nome;
		obra.dataCriacao = dataCriacao;
		obra.autor = autor;
		obra.classificacao = classificacao;
		obra.localProducao = localProducao;
		obra.descricao = descricao;
		return obra;
	}

	public void editarCadastroObra(long codigo, String nome, DateTime dataCriacao,
String autor, String classificacao, String localProducao, String descricao)
	{
		Obra obra = new Obra(codigo);
		obra.nome = nome;
		obra.dataCriacao = dataCriacao;
		obra.autor = autor;
		obra.classificacao = classificacao;
		obra.localProducao = localProducao;
		obra.descricao = descricao;
	}

	public IList<Obra> consultarAcervo(String nomeAcervo)
	{
		// [[Implementação aqui]]
		Acervo acervo = new Acervo(nomeAcervo);
		return acervo.listarObras();
	}
}

public class GerenciarEvento
{
	public Evento agendarEvento(String nomeEvento, DateTime inicio, DateTime
encerramento, String descricaoEvento, String textoConvites)
	{
		Evento evento = new Evento();
		evento.nomeEvento = nomeEvento;
		evento.inicio = inicio;
		evento.encerramento = encerramento;
		evento.descricaoEvento = descricaoEvento;
		evento.textoConvites = textoConvites;
		return evento;
	}
```

```
	public void editarEvento(int codigoEvento, String nomeEvento, DateTime inicio,
DateTime encerramento, String descricaoEvento, String textoConvites)
	{
		Evento evento = new Evento(codigoEvento);
		evento.nomeEvento = nomeEvento;
		evento.inicio = inicio;
		evento.encerramento = encerramento;
		evento.descricaoEvento = descricaoEvento;
		evento.textoConvites = textoConvites;
	}

	public void cancelarEvento(int codigoEvento)
	{
		Evento evento = new Evento(codigoEvento);
		evento.cancelar();
	}

	public void enviarConvites(int codigoEvento, List<String> cpfVisitantes)
	{
		Evento evento = new Evento(codigoEvento);
		foreach (var cpf in cpfVisitantes) {
			Visitante visitante = new Visitante(cpf);
			evento.convidar(visitante);
		}
	}

	public void reservarSala(int codigoEvento, int numeroSala)
	{
		Evento evento = new Evento(codigoEvento);
		Sala sala = new Sala(numeroSala);
		evento.sala = sala;
	}
}

public class GerenciarExposicoes
{
	public Exposicao cadastrarExposicao(String nomeExposicao, String descExposicao)
	{
		Exposicao exposicao = new Exposicao();
		exposicao.nomeExposicao = nomeExposicao;
		exposicao.descExposicao = descExposicao;
		return exposicao;
	}

	public void addObraEmExposicao(long codigoObra, long codigoExposicao)
	{
		Exposicao exposicao = new Exposicao(codigoExposicao);
		exposicao.obras.Add(new Obra(codigoObra));
	}

	public void editarExposicao(long codExposicao, String nomeExposicao, String
descExposicao)
	{
		Exposicao exposicao = new Exposicao(codExposicao);
		exposicao.nomeExposicao = nomeExposicao;
		exposicao.descExposicao = descExposicao;
```

```
	}

	public IList<Exposicao> consultarExposicoes()
	{
		// [[Implementação aqui]]
		return new List<Exposicao>{new Exposicao()}; // Nao copiar, avoid compile
error
	}

	public void retirarObraEmExposicao(long codigoObra, long codigoExposicao)
	{
		Exposicao exposicao = new Exposicao(codigoExposicao);
		exposicao.obras.Remove(new Obra(codigoObra));
	}

	public IList<Obra> consultarObrasEmExposicao(Exposicao exposicao)
	{
		// [[Implementação aqui]]
		return new List<Obra>{new Obra()}; // Nao copiar, avoid compile error
	}

	public void reservarSala(long codExposicao, int numeroSala)
	{
		Exposicao exposicao = new Exposicao(codExposicao);
		Sala sala = new Sala(numeroSala);
		exposicao.sala = sala;
	}
}

public class GerenciarLoja
{
	public Produto cadastrarProduto(String nomeProduto, String descricaoProduto, int
qtdProduto, float precoDeCusto, float precoDeVenda)
	{
		Produto produto = new Produto();
		produto.nomeProduto = nomeProduto;
		produto.descricaoProduto = descricaoProduto;
		produto.qtdProduto = qtdProduto;
		produto.precoDeCusto = precoDeCusto;
		produto.precoDeVenda = precoDeVenda;
		return produto;
	}

	public void editarProduto(long codProduto, String nomeProduto, String
descricaoProduto, int qtdProduto, float precoDeCusto, float precoDeVenda)
	{
		Produto produto = new Produto(codProduto);
		produto.nomeProduto = nomeProduto;
		produto.descricaoProduto = descricaoProduto;
		produto.qtdProduto = qtdProduto;
		produto.precoDeCusto = precoDeCusto;
		produto.precoDeVenda = precoDeVenda;
	}

	public void comprarProduto(long codProduto, int qtd)
	{
```

```
            Produto produto = new Produto(codProduto);
            produto.qtdProduto += qtd;
      }

      public void baixaEstoque(long codProduto, int qtd)
      {
            Produto produto = new Produto(codProduto);
            produto.qtdProduto -= qtd;
      }

      public Fornecedor cadastrarFornecedor(String Nome, String CNPJ, String email,
long telefone)
      {
            Fornecedor fornecedor = new Fornecedor();
            fornecedor.Nome = Nome;
            fornecedor.CNPJ = CNPJ;
            fornecedor.email = email;
            fornecedor.telefone = telefone;
            return fornecedor;
      }

      public void addItem(int idPedido, long codProduto, int qtd)
      {
            Pedido pedido = new Pedido(idPedido);
            Produto produto = new Produto(codProduto);
            Item item = pedido.items.Where(item => item.codProduto ==
codProduto).FirstOrDefault();
            if(item == null)
            {
                  item = new Item();
                  item.codProduto = codProduto;
            }
            item.qtdItens += qtd;
      }

      public void finalizarPedido(int idPedido)
      {
            new Pedido(idPedido).finalizar();
      }

      public void gerarRelatorio()
      {
            // [[Implementação aqui]]
      }

      public void retirarItem(int idPedido, long codProduto, int qtd)
      {
            Pedido pedido = new Pedido(idPedido);
            Produto produto = new Produto(codProduto);
            Item item = pedido.items.Where(item => item.codProduto ==
codProduto).First();
            item.qtdItens -= qtd;
            if(item.qtdItens < 0)
            {
                  pedido.items.Remove(item);
            }
```

```
	}
}

public class GerenciarRestauracoes
{
	public Restauracao solicitarRestauracao(long codigoObra, long
numCrachaRestaurador, float valorRestauracao, String detalhesRestauracao)
	{
		Restaurador restaurador = new Restaurador(numCrachaRestaurador);
		Obra obra = new Obra(codigoObra);
		Restauracao restauracao = new Restauracao();
		restauracao.obraRestaurada = obra;
		restauracao.restauradorResponsavel = restaurador;
		restauracao.valorRestauracao = valorRestauracao;
		restauracao.detalhesRestauracao = detalhesRestauracao;
		return restauracao;
	}

	public void atualizarStatusRestauracao(String status, long idRestauracao)
	{
		Restauracao restauracao = new Restauracao(idRestauracao);
		restauracao.status = status;
	}

	public void finalizarRestauracao(long idRestauracao)
	{
		Restauracao restauracao = new Restauracao(idRestauracao);
		restauracao.dataFimRestauracao = new DateTime();
		restauracao.status = "Finalizada";
	}

	public IList<Restauracao> consultarRestauracoes()
	{
		// [[Implementação aqui]]
	}

	public IList<Restaurador> consultarRestauradores()
	{
		// [[Implementação aqui]]
	}
}

public class GerenciarVisita
{
	public Visitante registrarVisitante(String email, String nome, String cpf,
String telefone)
	{
		Visitante visitante = new Visitante();
		visitante.email = email;
		visitante.nome = nome;
		visitante.cpf = cpf;
		visitante.telefone = telefone;
		return visitante;
	}
```

```
	public Visita iniciarVisita(DateTime dataHora, int idCartao, float valorEntrada,
String cpfVisitante)
	{
		Visita visita = new Visita();
		visita.dataHoraEntrada = dataHora;
		visita.idCartao = idCartao;
		visita.valorEntrada = valorEntrada;
		visita.visitante = new Visitante(cpfVisitante);
		return visita;
	}

	public void editarRegistroVisitante(String email, String nome, String cpf,
String telefone)
	{
		Visitante visitante = new Visitante(cpf);
		visitante.email = email;
		visitante.nome = nome;
		visitante.telefone = telefone;
	}

	public void encerrarVisita(int idCartao)
	{
		Visita visita = new Visita(idCartao);
		visita.dataHoraSaida = new DateTime();
	}

	public int emitirCartao(String cpfVisitante)
	{
		Visitante visitante = new Visitante();
		return visitante.gerarIdCartao();
	}

	public IList<Visitante> consultarVisitantes()
	{
		// [[Implementação aqui]]
	}
}
```

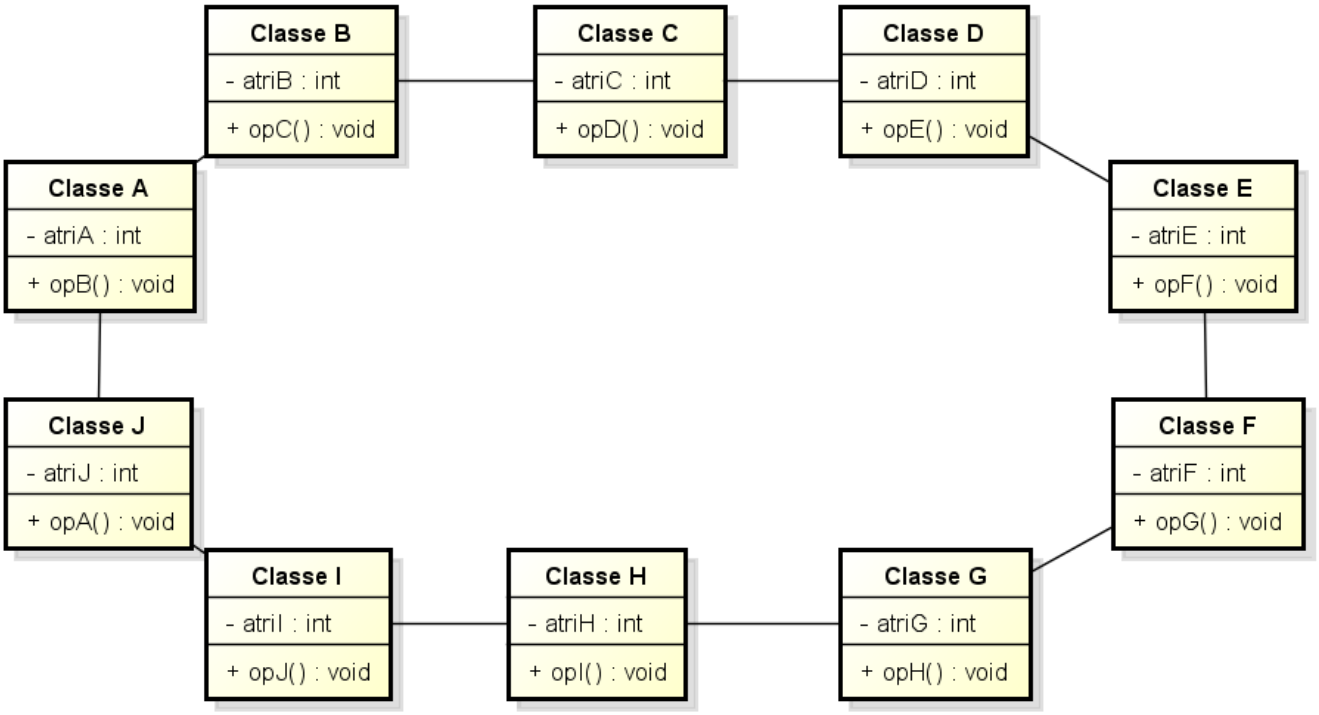
7. ATIVIDADE DE ABSTRAÇÃO

O diagrama da atividade apresenta tanto dependência estrutural como dependência não estrutural e baixa coesão, pois todas as classes apresentam atributos e métodos que não correspondem às respectivas responsabilidades da classe em questão.

Exemplo: A classe A contem como atributo “atriB” que se refere a classe B e contém o método “opC” que se refere a classe C.

Por consequência da baixa coesão, o diagrama apresenta alto índice de acoplamento, apresentando relacionamentos que não existiriam caso todas as classes fossem mais coesas.

Exemplo: A Classe A tem associação com as classes B, J, I e C, pois essa classe contém atributo da Classe B um método da Classe C e a Classe J contem atributo de A e a Classe I um método da Classe A.



No novo diagrama, as associações são feitas por meio de dependência estrutural, os atributos das classes correspondem a sua respectiva classe, aumentando a coesão (Classe A contém atributo atriA) e cada classe faz somente dois relacionamentos, diminuindo o acoplamento.

PS: Os relacionamentos estruturais não estão explícitos por esse ser um Diagrama de Classes de Análise e não de Projeto.