Engenharia de Software III

Lista 1

Membros:

Nome	RA
Gabriel Ferrari Dantas	1680481511015
Joel Braga Barreto	1680481611043
Roberto Zaniboni de Souza	1680481621001
Matheus Faria Landim	1680481711019
Juan laneiro de souza	1680481511025
Mateus Costa	1680141682399

1.

RF01: Gestão de Visita

Descrição: Atendente faz o possível cadastro de visitantes.

Atores: Atendente.

Fluxo Principal:

1. Atendente seleciona a opção “Verificar Cadastro”.
2. Atendente entrega o cartão magnético do visitante.
3. Fim Caso de uso.

Fluxo Alternativo:

(A1) Alternativa ao passo 1.- Visitante não cadastrado.

1. Atendente seleciona a opção “Cadastrar Visitante”.
2. Atendente insere o nome do visitante.
3. Atendente insere qual é o “Tipo de Visitante” : “Palestrante” , “Visitante comum” , “Responsável pelo evento”.
4. Atendente insere o telefone do visitante.
5. Atendente insere o endereço do visitante.
6. Atendente insere o rg do visitante.
7. Atendente seleciona opção “Salvar Visitante”.
8. O sistema efetua o cadastro do visitante.
9. Caso de uso é encerrado.

(A2) Alternativa ao passo 5. – Visitante já cadastrado.

1. O sistema exibe a mensagem de que o visitante já está cadastrado.
2. Caso de uso é encerrado.

Fluxo de Exceção:

(E1) Exceção do item 3 no Fluxo alternativo 1(A1). – Campo “Tipo de Visitante” vazio.

1. O sistema exibe uma mensagem de que o campo “Tipo de Visitante” está vazio.
2. Retorna ao fluxo alternativo (A1).

2.

RF02: Gestão de Acervo

Descrição: Atendente faz o cadastro das obras ou documentos históricos existentes no museu.

Atores: Atendente.

Fluxo Principal:

1. Atendente seleciona a opção “Cadastrar Obras/Documentos”.
2. Atendente informa título da obra ou documento.
3. Atendente informa o autor da obra ou documento.
4. Atendente informa a data da obra ou documento.
5. Atendente informa o estado “Ótimo” da obra ou documento.
6. Atendente informa o tipo da Obra entre “Documento” , “Quadro” , “Estátua”.
7. Atendente seleciona a opção “Salvar”.
8. O sistema efetua o cadastro da obra ou documento.
9. Caso de uso é encerrado.

Fluxo Alternativo:

(A1) Alternativa ao passo 5. – Recomendável restauração.

1. Caso a opção selecionada seja diferente de “Ótimo” ou “Bom” exibe uma mensagem de restauração recomendável.
2. Retorna ao fluxo principal do sistema.

3.

RF03: Gestão de Exposição

Descrição: Gerente faz o cadastro das obras ou documentos históricos presentes em determinada exposição.

Atores: Gerente.

Fluxo Principal:

1. Gerente seleciona a opção “Exposições”
2. Gerente seleciona “Criar exposição”
3. Gerente insere o nome da exposição.
4. Gerente insere as obras presentes naquela exposição.
5. Gerente insere a quantidade máxima de visitantes na exposição.
6. Gerente insere em qual sala será reservada para a exposição.
7. Gerente seleciona se a exposição é temporária ou não.
8. Gerente seleciona a opção “não”.
9. O sistema efetua o cadastro da exposição.
10. Caso de uso é encerrado.

Fluxo Alternativo:

(A1) Alternativo ao passo 7. – Exposição temporária.

1. O Gerente informa que a exposição é temporária.
2. O Gerente informa qual a data de início da exposição.
3. O Gerente informa qual a data de encerramento da exposição.
4. Retorna ao fluxo principal do sistema.

4.

RF04: Gestão de Restauração

Descrição: Gerente controla a restauração das obras.

Atores: Gerente.

Fluxo Principal:

1. Gerente seleciona a opção “Restaurar obras”.
2. Gerente informa qual obra deve ser restaurada.
3. Gerente informa qual empresa irá restaurar a obra ou documento.
4. Gerente informa as datas limites para o retorno da obra.
5. Gerente seleciona a opção “Solicitar Restauração”
6. O sistema cadastra a obra em restauração e a torna indisponível para exposições.
7. Fim caso de uso.

Fluxo Alternativo:

(A1) Alternativo ao passo 2. – Obra já está sendo restaurada.

5. O sistema informa que a obra já está sendo restaurada.
6. O sistema retorna a tela anterior.

5.

RF05: Gestão de Evento

Descrição: Gerente controla os eventos que estão sendo executados no museu.

Atores: Gerente.

Fluxo Principal:

1. Gerente seleciona a opção “Eventos”.
2. Gerente seleciona a opção “Criar eventos”.
3. Gerente informa o nome do evento.
4. Gerente informa a descrição do evento.
5. Gerente informa o responsável por aquele evento.
6. Gerente informa qual será a data do evento.
7. Gerente informa a quantidade de visitantes para aquele evento.
8. Gerente seleciona a opção “Salvar Evento”
9. O sistema cria e salva o evento.
10. Fim caso de uso.

Fluxo Alternativo:

(A1) Alternativo ao item 5. – Responsável não cadastrado.

1. O sistema exibe uma mensagem informando que o responsável não está cadastrado em “Visitante”.
2. Retorna ao fluxo principal.

(A2) Alternativo ao item 6. – Data inválida.

1. O sistema exibe uma mensagem de data inválida.

2. Retorna ao fluxo principal.

6.

RF06: Gestão de Venda de Souvenir

Descrição: Gerente controla a loja de souvenir e o estoque da mesma.

Atores: Gerente.

Fluxo Principal:

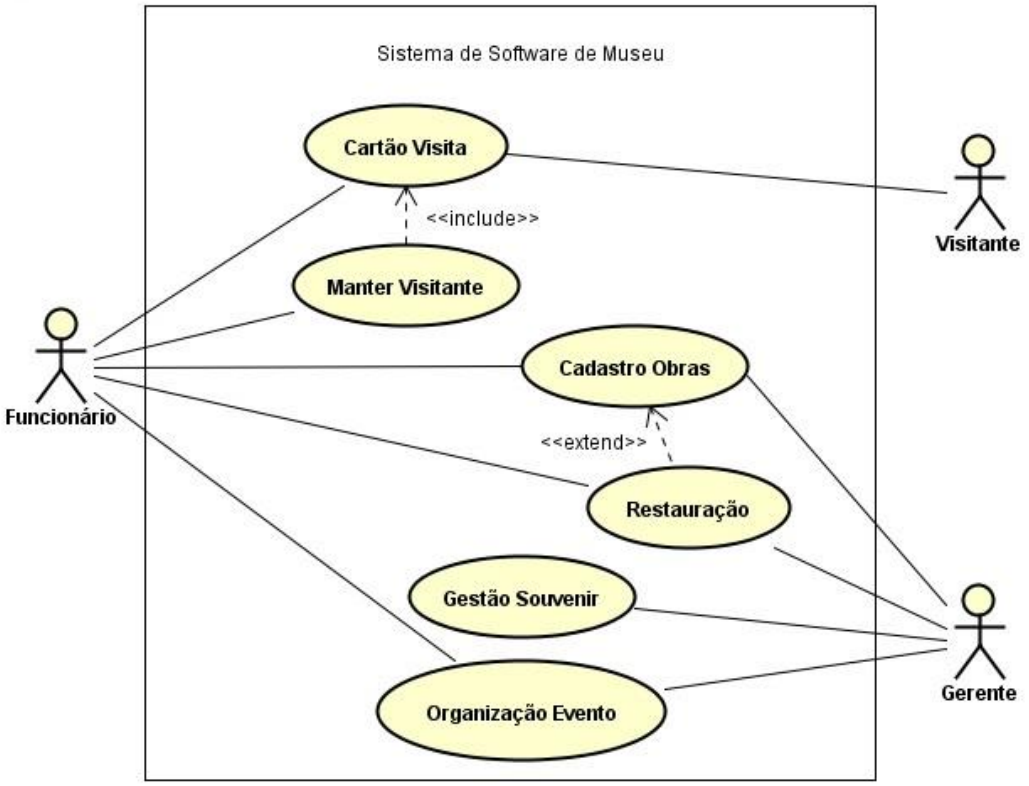
1. Gerente seleciona a opção “Loja de Venda”.
2. Gerente seleciona a opção “Estoque”.
3. Gerente verifica a quantidade do produtos no estoque.
4. Gerente informa o produto a ser comprado.
5. Gerente informa a quantidade que será comprada.
6. O sistema salva o novo pedido.
7. Fim caso de uso.

Fluxo de Alternativo :

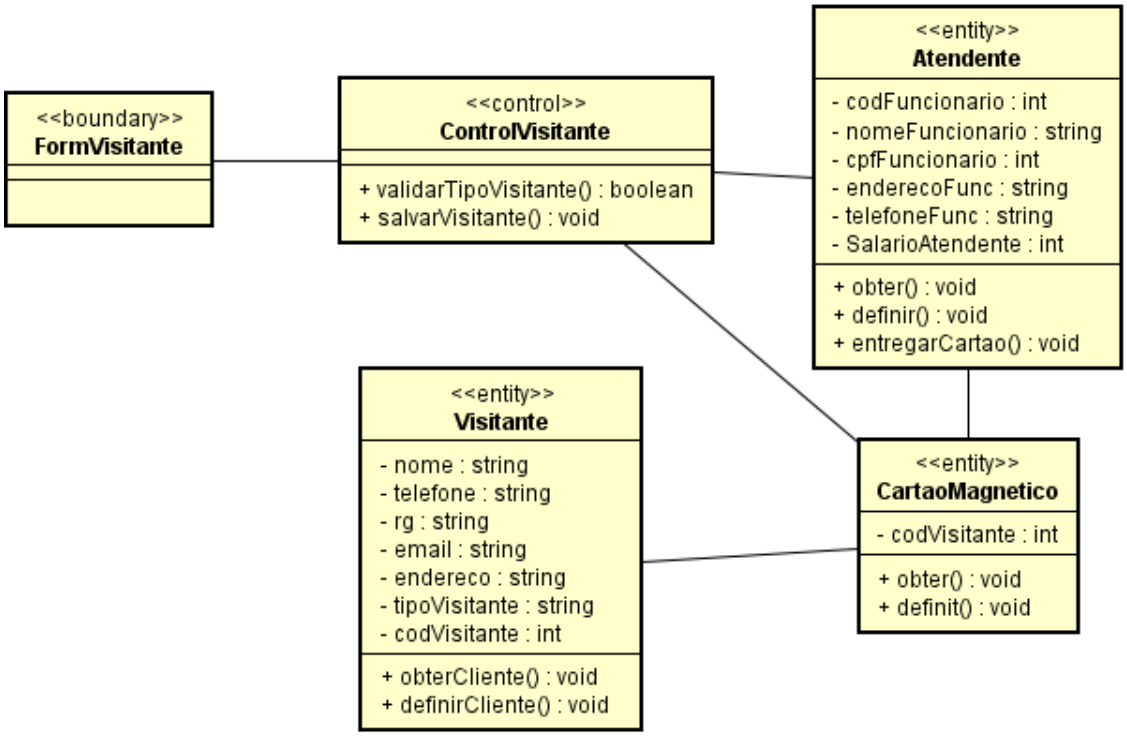
(A1) Alternativo ao passo 2. – Seleciona relatórios.

1. Gerente seleciona a opção “Relatório Mensal”.
2. Fim caso de uso.

7.


Sistema de Software de Museu
Cartão Visita
<<include>>
Manter Visitante
Cadastro Obras
<<extend>>
Restauração
Gestão Souvenir
Organização Evento
Visitante
Funcionário
Gerente

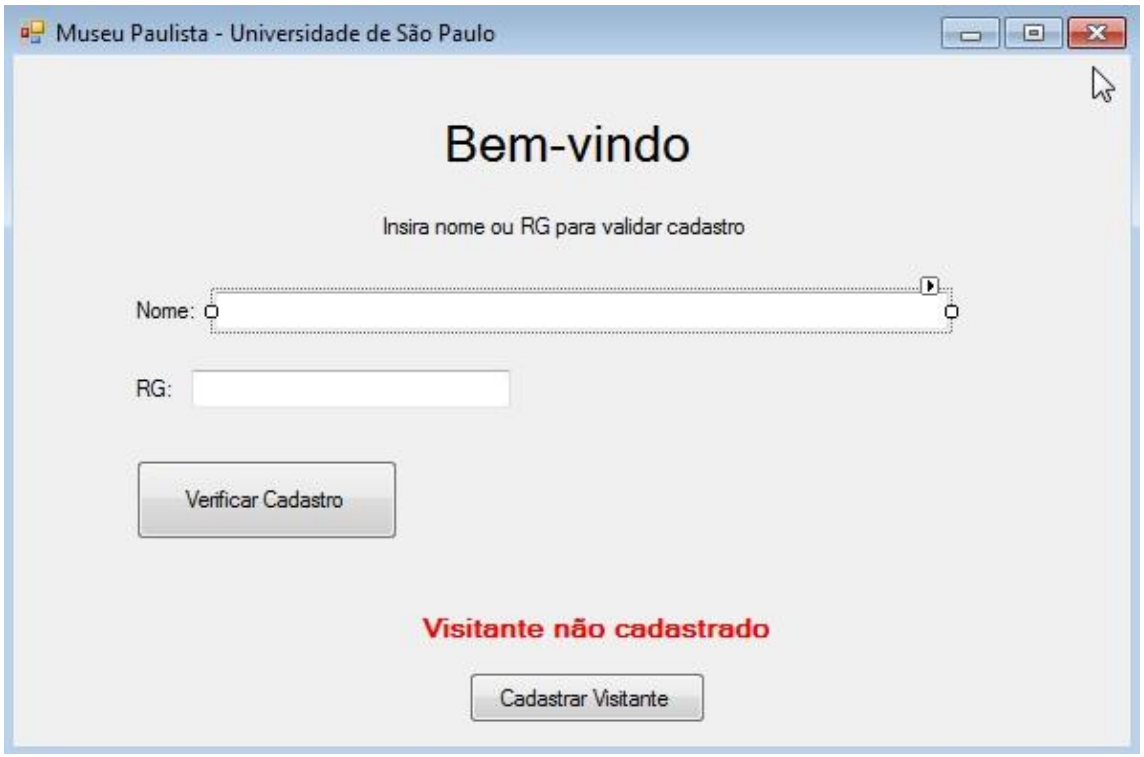
8.

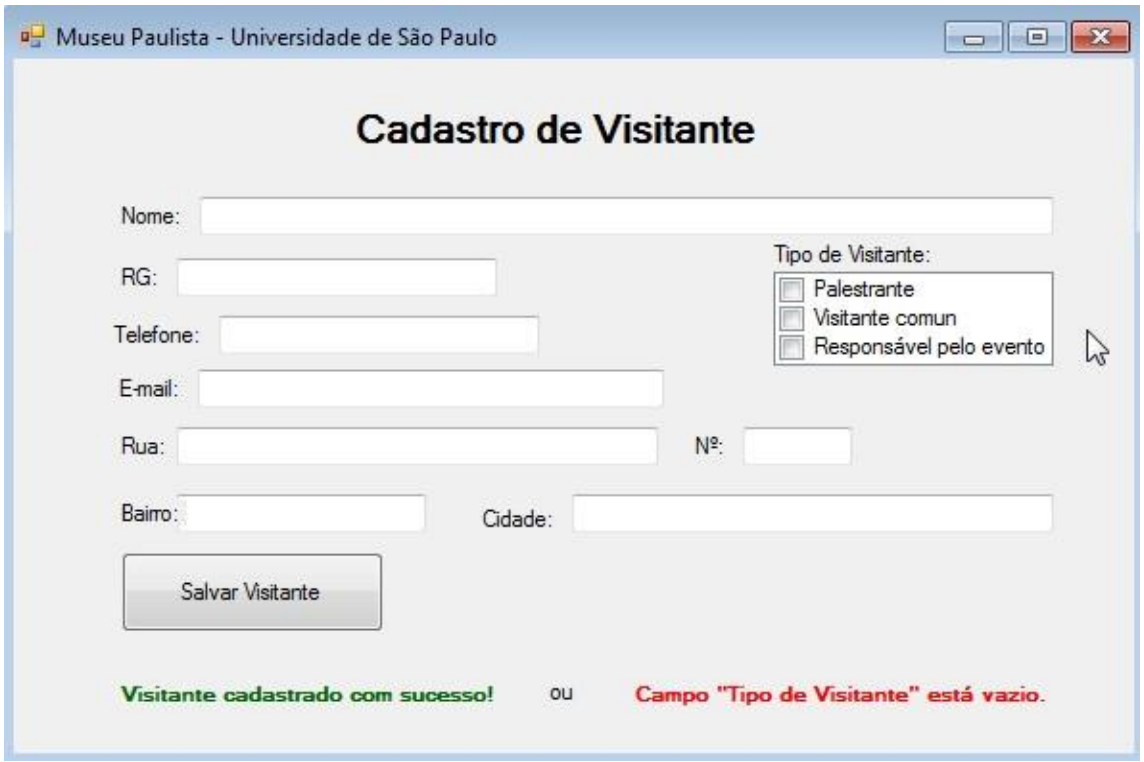

<<boundary>>
FormVisitante
<<control>>
ControlVisitante
+ validarTipoVisitante() : boolean
+ salvarVisitante() : void
<<entity>>
Atendente
- codFuncionario : int
- nomeFuncionario : string
- cpfFuncionario : int
- enderecoFunc : string
- telefoneFunc : string
- SalarioAtendente : int
+ obter() : void
+ definir() : void
+ entregarCartao() : void
<<entity>>
Visitante
- nome : string
- telefone : string
- rg : string
- email : string
- endereco : string
- tipoVisitante : string
- codVisitante : int
+ obterCliente() : void
+ definirCliente() : void
<<entity>>
CartaoMagnetico
- codVisitante : int
+ obter() : void
+ definit() : void

8. INTERFACE

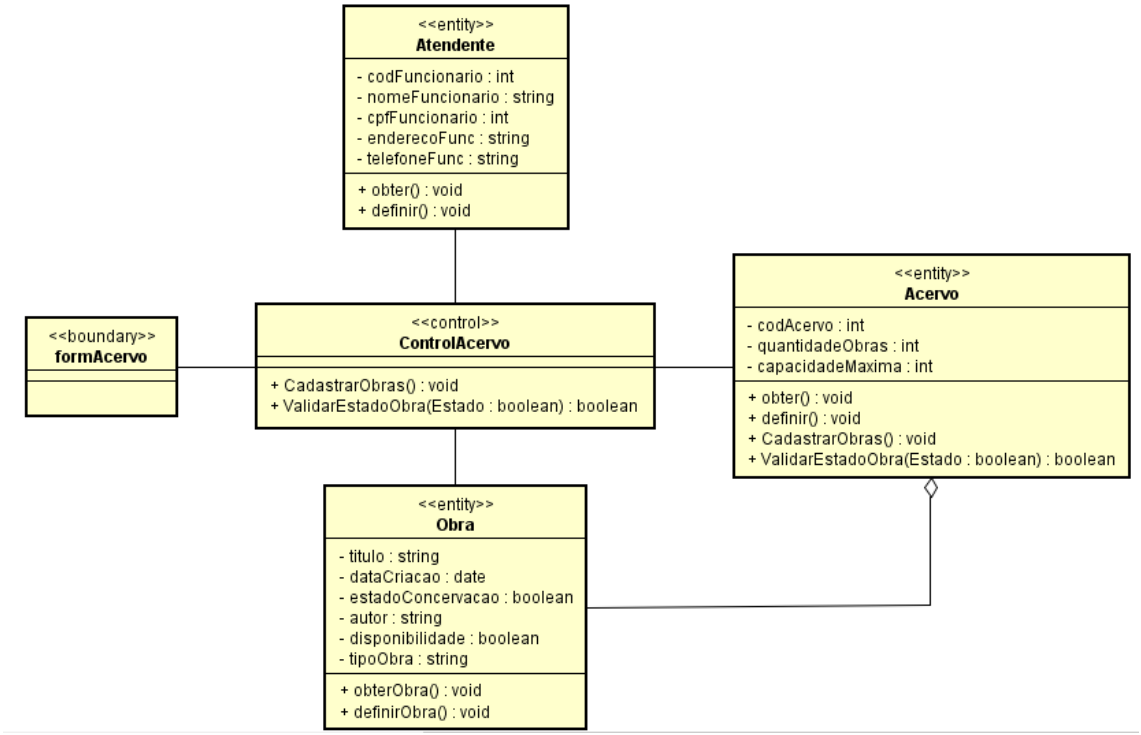

Museu Paulista - Universidade de São Paulo
MENU
Verificar Cadastro
Cadastrar Visitante
Cadastrar Obras/Documentos
Exposições
Restaurar Obras
Eventos
Loja de Venda


Museu Paulista - Universidade de São Paulo
Bem-vindo
Insira nome ou RG para validar cadastro
Nome:
RG:
Verificar Cadastro
Visitante cadastrado!


Museu Paulista - Universidade de São Paulo
Bem-vindo
Insira nome ou RG para validar cadastro
Nome:
RG:
Verificar Cadastro
Visitante não cadastrado
Cadastrar Visitante

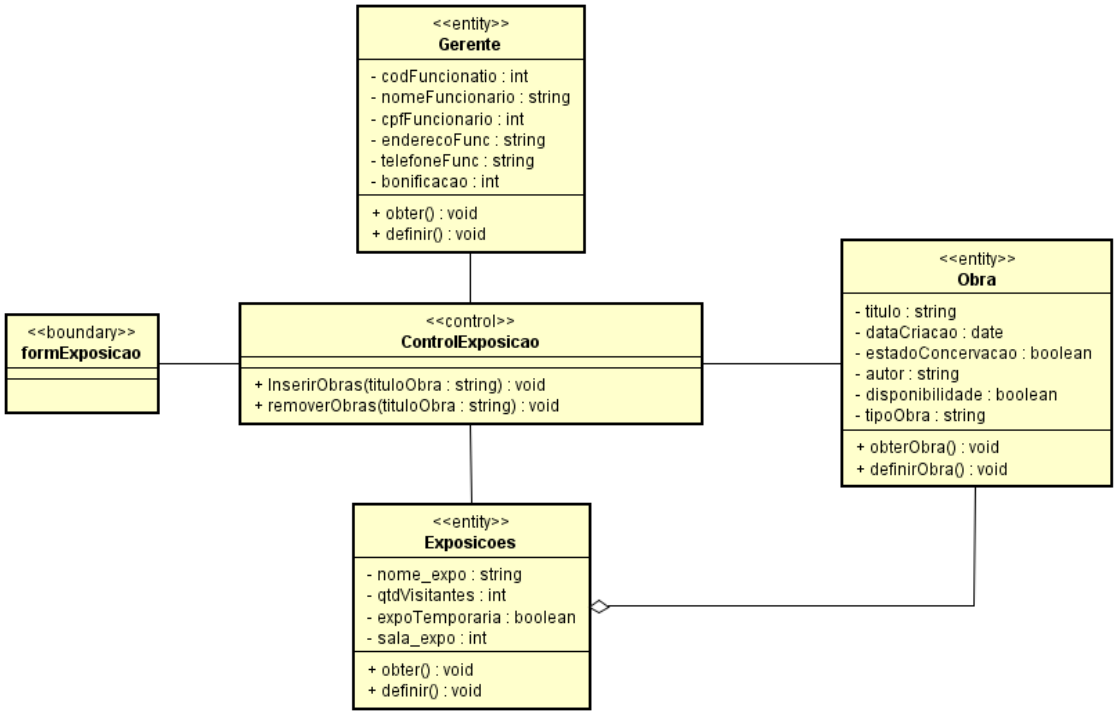

Museu Paulista - Universidade de São Paulo
Cadastro de Visitante
Nome:
RG:
Telefone:
E-mail:
Rua:
Nº:
Bairro:
Cidade:
Tipo de Visitante:
Palestrante
Visitante comun
Responsável pelo evento
Salvar Visitante
Visitante cadastrado com sucesso!
ou
Campo "Tipo de Visitante" está vazio.

9.

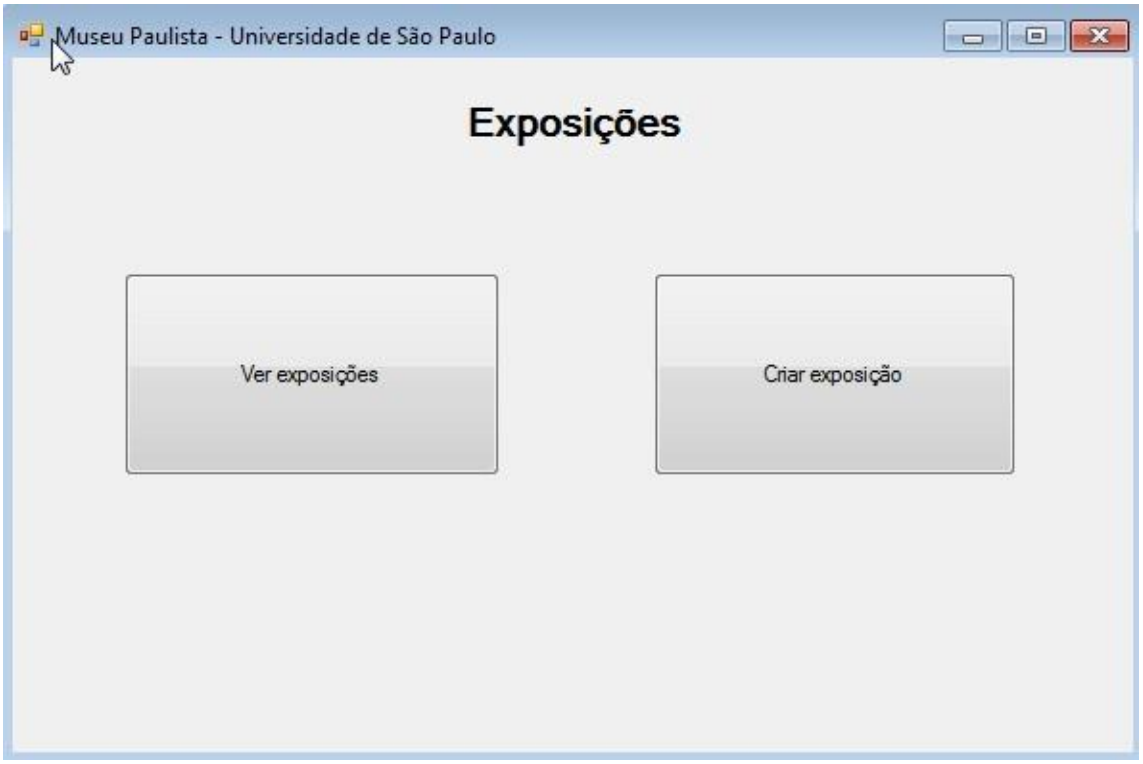


9. INTERFACE

10.

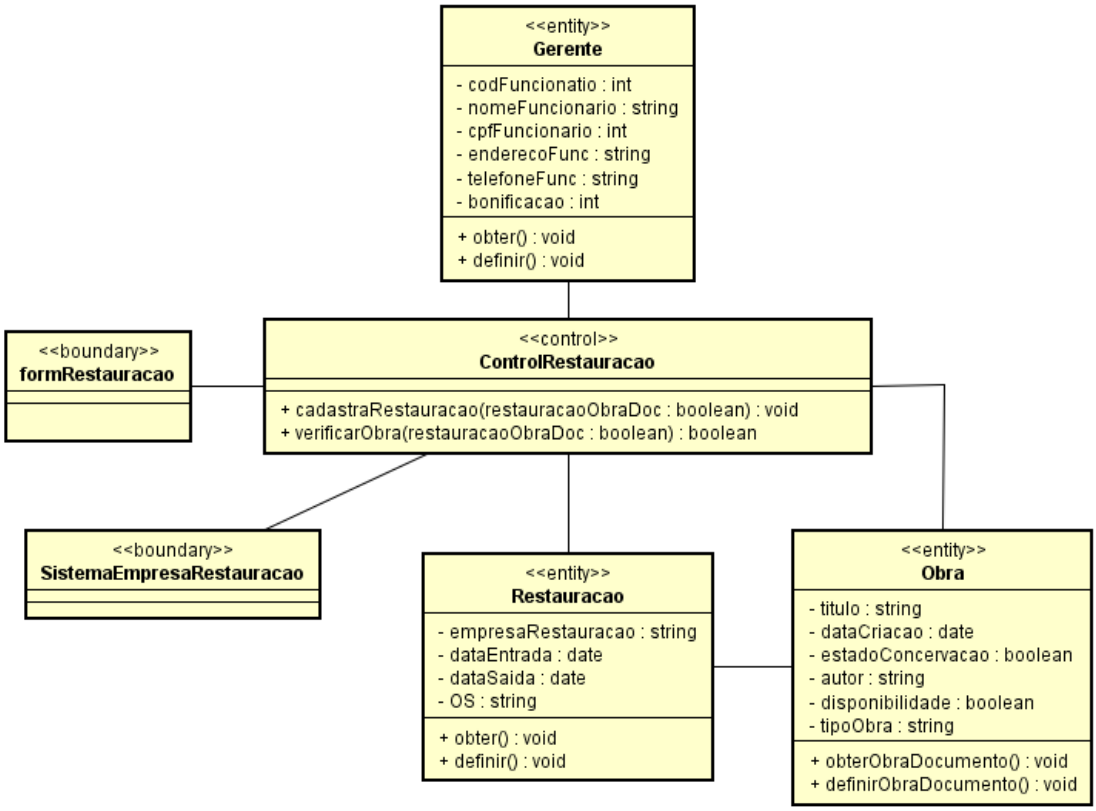


10. INTERFACE



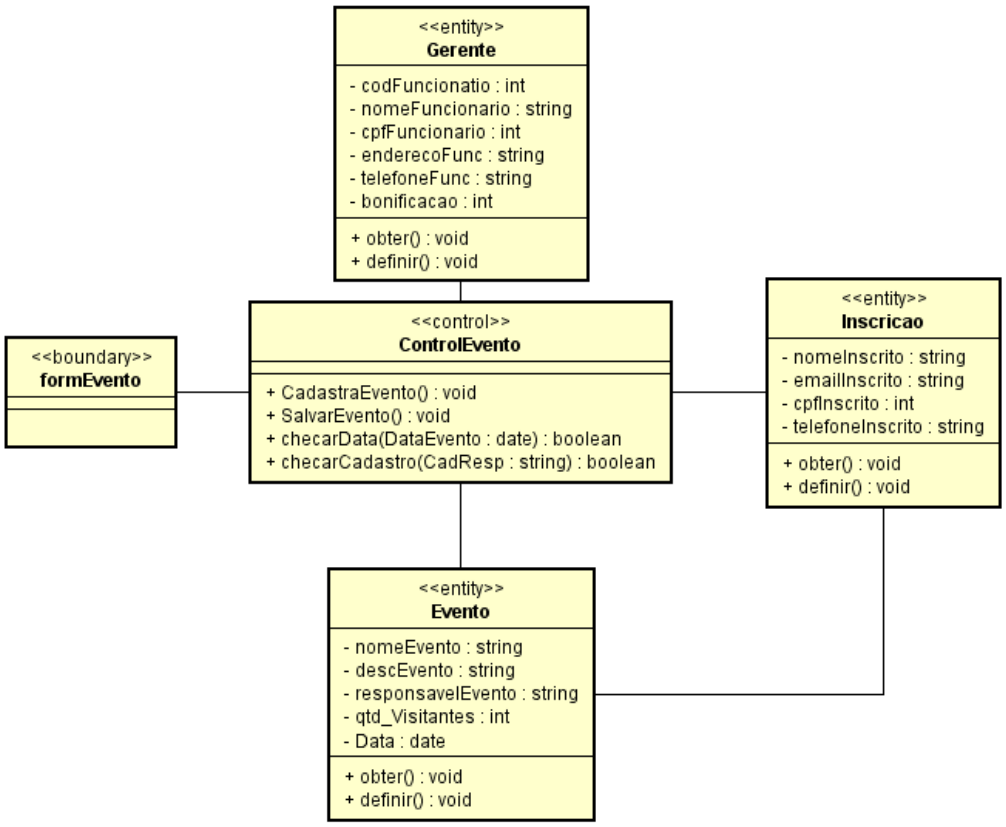

Museu Paulista - Universidade de São Paulo
Criar exposições
Nome:
Obras:
Qtda máxima:
Sala:
Salvar
Temporária:
Sim
Não
Data início:
Data Fim:
Exposição criada!
ou
Existe algum campo "vazio"

11.

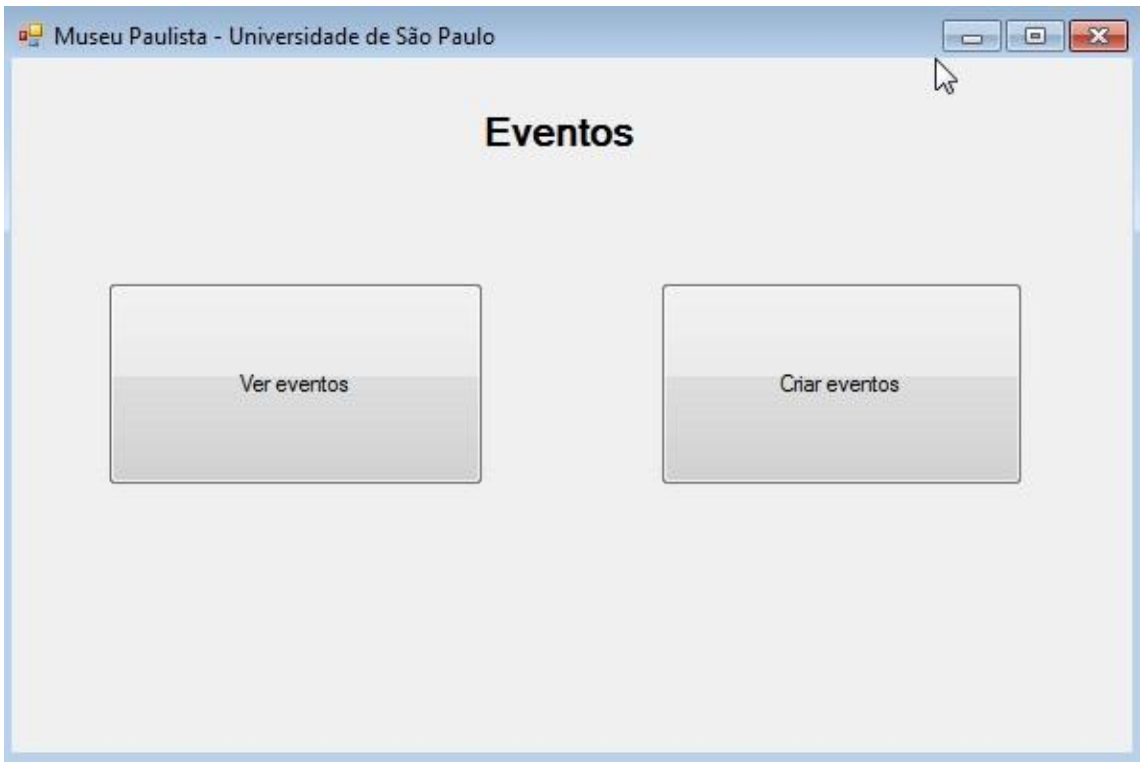

<<entity>>
Gerente
- codFuncionatio : int
- nomeFuncionario : string
- cpfFuncionario : int
- enderecoFunc : string
- telefoneFunc : string
- bonificacao : int
+ obter() : void
+ definir() : void
<<boundary>>
formRestauracao
<<control>>
ControlRestauracao
+ cadastraRestauracao(restauracaoObraDoc : boolean) : void
+ verificarObra(restauracaoObraDoc : boolean) : boolean
<<boundary>>
SistemaEmpresaRestauracao
<<entity>>
Restauracao
- empresaRestauracao : string
- dataEntrada : date
- dataSaida : date
- OS : string
+ obter() : void
+ definir() : void
<<entity>>
Obra
- titulo : string
- dataCriacao : date
- estadoConcervacao : boolean
- autor : string
- disponibilidade : boolean
- tipoObra : string
+ obterObraDocumento() : void
+ definirObraDocumento() : void

11. INTERFACE

12.

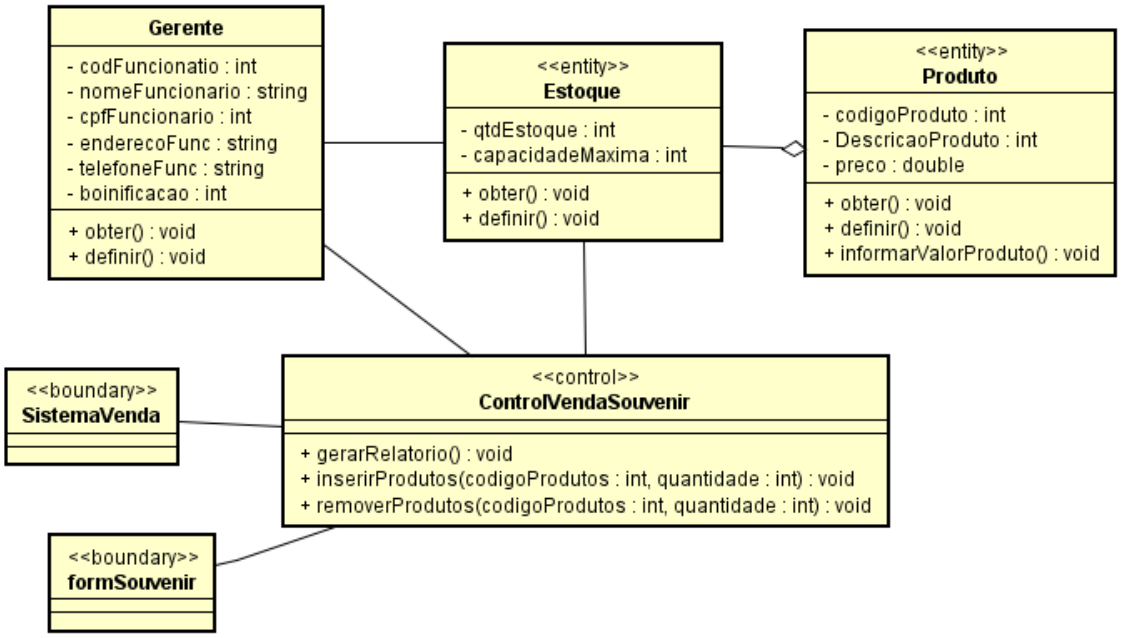


12. INTERFACE



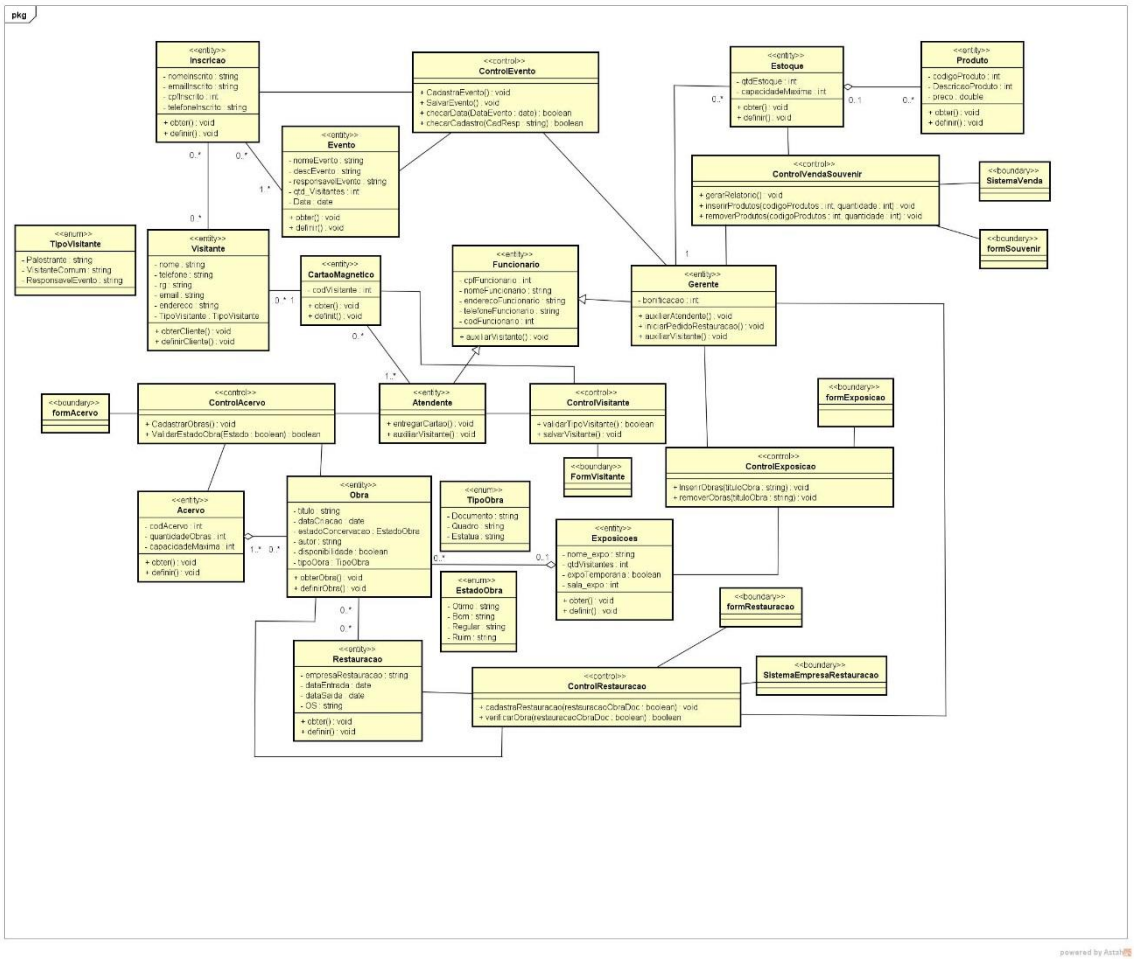

Museu Paulista - Universidade de São Paulo
Criar eventos
Nome:
Descrição:
Responsável:
Data:
Qtda. Visitantes:
Salvar Evento
Evento criado com sucesso!
ou
Responsável não está cadastrado em "Visitante"
Data Inválida

13.


Gerente
- codFuncionatio : int
- nomeFuncionario : string
- cpfFuncionario : int
- enderecoFunc : string
- telefoneFunc : string
- boinificacao : int
+ obter() : void
+ definir() : void
<<entity>>
Estoque
- qtdEstoque : int
- capacidadeMaxima : int
+ obter() : void
+ definir() : void
<<entity>>
Produto
- codigoProduto : int
- DescricaoProduto : int
- preco : double
+ obter() : void
+ definir() : void
+ informarValorProduto() : void
<<boundary>>
SistemaVenda
<<control>>
ControlVendaSouvenir
+ gerarRelatorio() : void
+ inserirProdutos(codigoProdutos : int, quantidade : int) : void
+ removerProdutos(codigoProdutos : int, quantidade : int) : void
<<boundary>>
formSouvenir

13. INTERFACE

14.



Arquivo do Astah em anexo no e-mail para melhor visualização.

15.

A classe mais coesa identificada pelo grupo foi a classe de entidade “Exposição”, pois está de acordo com o princípio de responsabilidade única, ou seja, a classe possui apenas uma única responsabilidade e realiza de maneira satisfatória.

A classe “Acervo” foi identificada pelo grupo como a classe menos coesa pois além de fazer a parte de cadastro das obras, ela precisa assumir outras responsabilidade além do contexto “Acervo”, como “Verificar Estado Obra”, para que facilitasse na hora da restauração em qual obra será restaurada.

16.

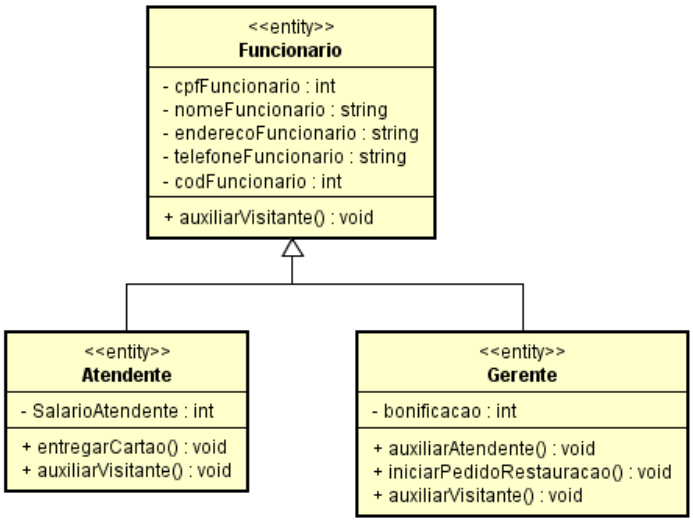
A classe identificada pelo grupo como mais acoplada pode ser considerada tanto a classe “Obra” pois ela tem uma dependência maior em

relação as outras, ou seja, uma mudança no sistema que envolva essas duas classes será mais complicada tal manutenção

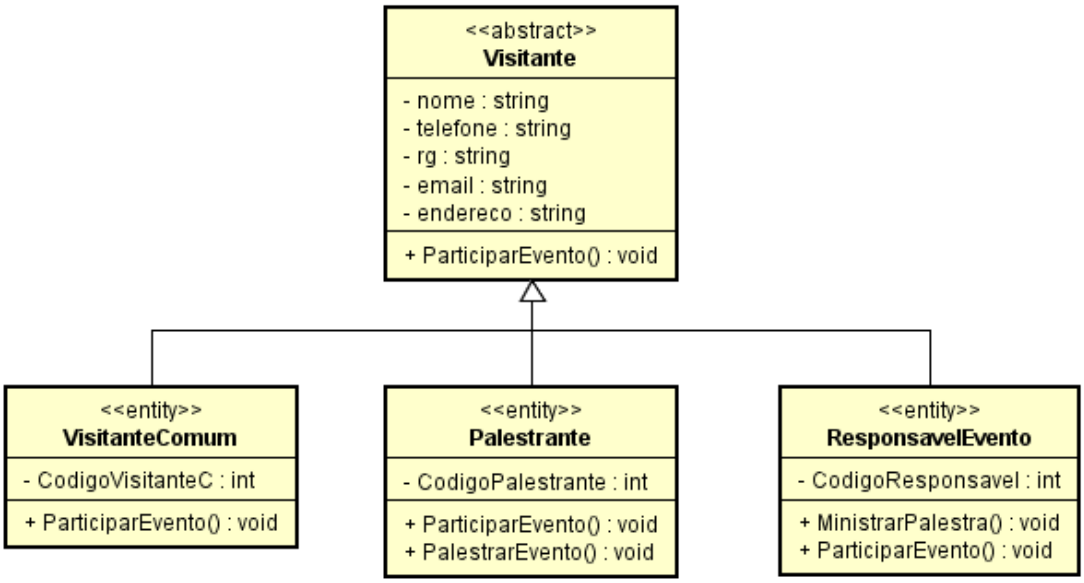
A classe identificada com baixo acoplamento foi a “Produto”, pois ela possui independência das classes para executar sua tarefa.

17.

Funcionário: A superclasse Funcionário é a generalização de dois tipos de funcionários Gerente ou Atendente que possuem funções diferente porem ambos sendo um tipo de funcionário dentro do museu.



Visitante: A superclasse Visitante é a generalização de três tipos de Visitante, o visitante comum, o palestrante e o Responsável pelo evento, porém tendo responsabilidades diferentes no museu.



As Gen/Espec acima não violam o Princípio de Liskov pois cada classe filha pode substituir completamente sua classe base(mãe). O que é possível, já que um Visitante Comum é um Visitante, como um Palestrante e um responsável pelo evento também são um visitante. Assim como Gerente é um funcionário e Atendente é um funcionário.

18. Estrutura em Java

Funcionário:

```
public class Funcionario {

        private int cpfFuncionario;
        private string nomeFuncionario;
        private string enderecoFuncionario;
        private string telefoneFuncionario;
        private int codFuncionario;

        public void auxiliarVisitante() {

        }
```

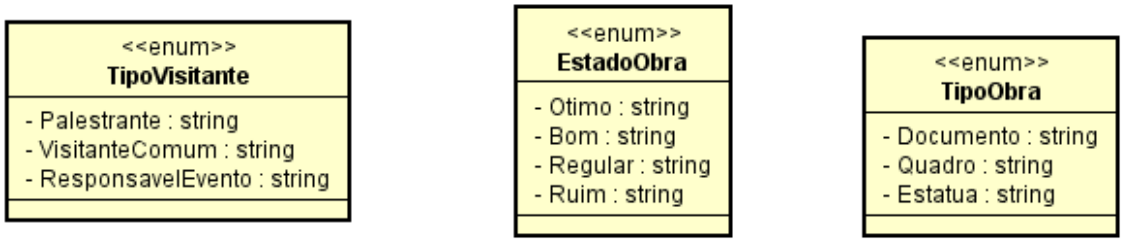
```
public class Gerente extends Funcionario {

	private int bonificacao;

	public void auxiliarAtendente() {

	}

	public void iniciarPedidoRestauracao() {

	}
	public void auxiliarVisitante() {
	}

}
public class Atendente extends Funcionario {

	private int SalarioAtendente;

	public void entregarCartao() {

	}
	public void auxiliarVisitante() {
	}

}
```

Visitante:

```
public class Visitante {

	private string nome;
	private string telefone;
	private string rg;
	private string email;
	private string endereco;

	public void ParticiparEvento() {
	}

}
public class VisitanteComum extends Visitante {

	private int CodigoVisitanteC;
	public void ParticiparEvento() {
	}

}

public class Palestrante extends Visitante {

	private int CodigoPalestrante;

	public void ParticiparEvento() {

	}
	public void PalestrarEvento() {
	}
```

```
}
public class ResponsavelEvento extends Visitante {

        private int CodigoResponsavel;
        public void MinistrarPalestra() {

        }
        public void ParticiparEvento() {
        }
}
```

19.



A classe enumerada "tipoVisitante" foi criada para poder atender a diversidade dos tipos visitante que podem comparecer ao museu.

A classe enumerada "EstadoObra" foi criada para poder diferenciar o estado de conservação em que as obras estão para que possa facilitar na tomada de decisão em qual obra será restaurada.

A classe enumerada "TipoObra" foi necessária para que pudesse distinguir os tipos de obras em que podem existir no acervo do museu.

20.

Estado Obra:

```
public enum EstadoObra {

        private string Otimo;
        private string Bom;
        private string Regular;
        private string Ruim;

}
```

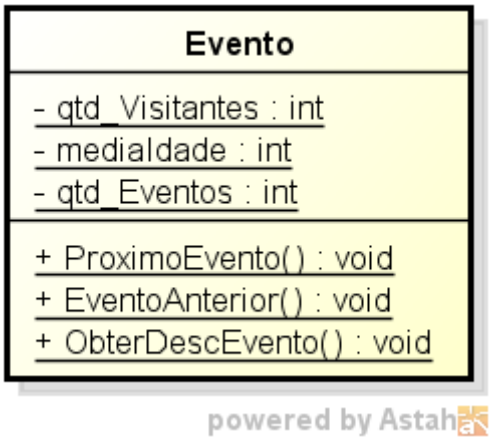
Tipo Obra:

```
public enum TipoObra {

        private string Documento;
        private string Quadro;
        private string Estatua;

}
```

Tipo Visitante:

```
public enum TipoVisitante {

        private string Palestrante;
        private string VisitanteComum;
        private string ResponsavelEvento;
}
```

21.



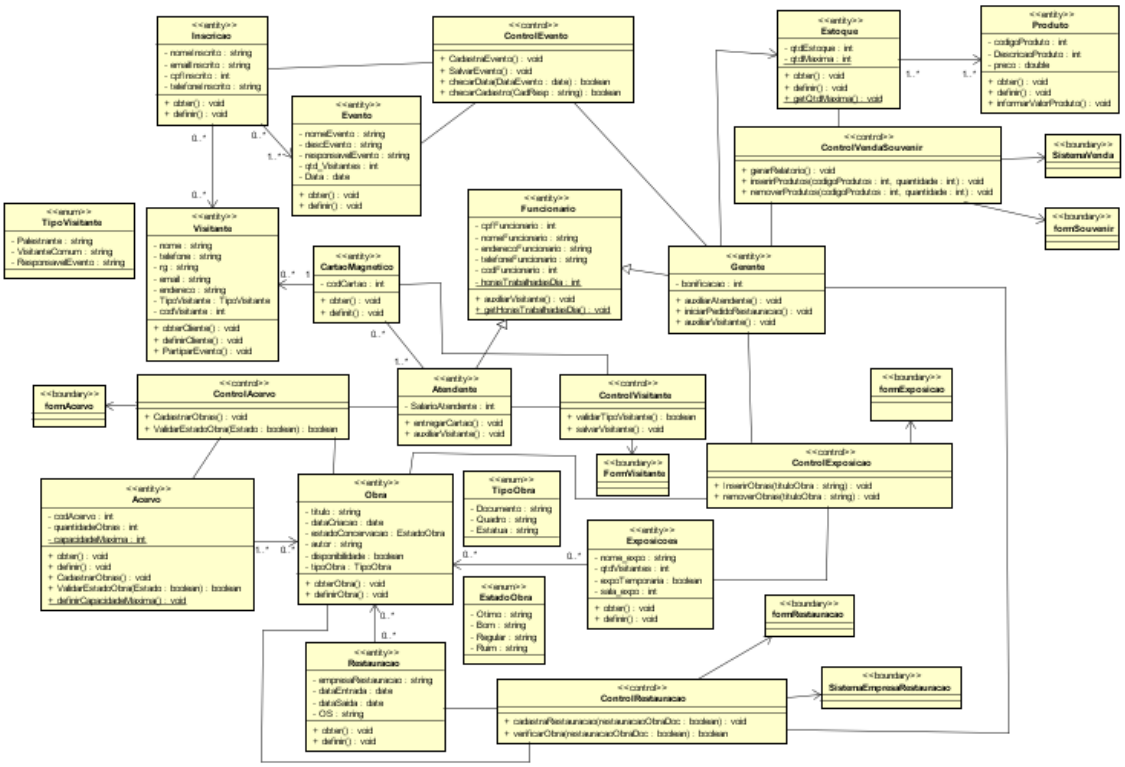
O atributo qtd_Visitantes server para ter o controle da quantidade de visitantes por evento. O atributo mediaIdade serve para obter dados sobre a faixa etária do público que está participando dos eventos. O atributo qtd_Eventos serve para ter o controle da quantidade de Eventos já realizados.

O método ProximoEvento serve para executar a adição de um novo Evento no sistema. O método EventoAnterior serve para consultar Eventos que já ocorreram. O método ObterDescEvento serve para conceder um desconto diferenciado dependendo do tipo do Visitante.

22.

```
public static class Visitante{

 private static int qtd_Visitantes;
 private static int mediaIdade;
 private static int qtd_Eventos;

 public static void ProximoEvento(){
   }
 public static void EventoAnterior(){
   }
 public static void ObterDescEvento(){
   }
}
```

23.



Vantagem: Aumento do desempenho

Desvantagem: Queda no encapsulamento porém aumento do acoplamento

24.

```
public class Acervo {
        private int codAcervo;
        private int quantidadeObras;
        private static int capacidadeMaxima;
        private Obra[] obra;
        public void obter() {}
        public void definir() {}
        public void CadastrarObras() {}
        public boolean ValidarEstadoObra(boolean Estado) {
                return false;}
        public static void definirCapacidadeMaxima() {}
}
```

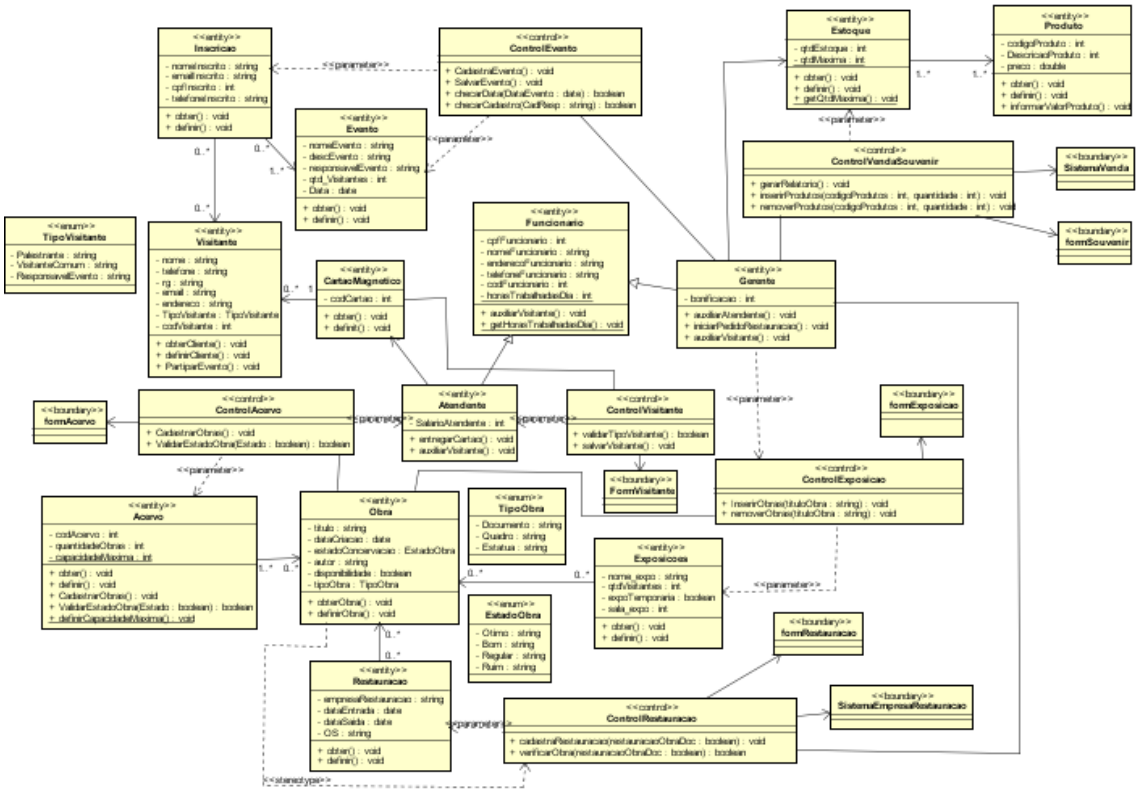
```
public class Atendente extends Funcionario {
	private int SalarioAtendente;
	private CartaoMagnetico cartaoMagnetico;
	public void entregarCartao() {}
	public void auxiliarVisitante() {}
}
public class Estoque {
	private int qtdEstoque;
	private static int qtdMaxima;
	private Produto[] produto;
	public void obter() {}
	public void definir() {}
	public static void getQtdMaxima() {}
}
public class Evento {

	private string nomeEvento;
	private string descEvento;
	private string responsavelEvento;
	private int qtd_Visitantes;
	private date Data;
	public void obter() {}
	public void definir() {	}
}
public class Exposicoes {

	private string nome_expo;
	private int qtdVisitantes;
	private boolean expoTemporaria;
	private int sala_expo;
```

```
        private Obra[] obra;
        public void obter() { }
        public void definir() {         }
}
public class Funcionario {

        private int cpfFuncionario;
        private string nomeFuncionario;
        private string enderecoFuncionario;
        private string telefoneFuncionario;
        private int codFuncionario;
        private static int horasTrabalhadasDia;
        public void auxiliarVisitante() {}
        public static void getHorasTrabalhadasDia() {  }
}
public class Gerente extends Funcionario {

        private int bonificacao;
        private Estoque estoque;
        public void auxiliarAtendente() {   }
        public void iniciarPedidoRestauracao() {}
        public void auxiliarVisitante() {     }
}
public class Inscricao {

        private string nomeInscrito;
        private string emailInscrito;
        private int cpfInscrito;
        private string telefoneInscrito;
        private Visitante[] visitante;
```

```
        private Evento[] evento;
        public void obter() { }
        public void definir() {         }
}
public class Obra {

        private string titulo;
        private date dataCriacao;
        private EstadoObra estadoConcervacao;
        private string autor;
        private boolean disponibilidade;
        private TipoObra tipoObra;
        public void obterObra() {    }
        public void definirObra() {  }
}
public class Produto {

        private int codigoProduto;
        private int DescricaoProduto;
        private double preco;
        public void obter() {}
        public void definir() {}
        public void informarValorProduto() {       }
}
public class Restauracao {

        private string empresaRestauracao;

        private date dataEntrada;
        private date dataSaida;
```

```
        private string OS;
        private Obra[] obra;
        public void obter() {}
        public void definir() {         }
}
public class Visitante {

        private string nome;
        private string telefone;
        private string rg;
        private string email;
        private string endereco;
        private TipoVisitante TipoVisitante;
        private int codVisitante;
        public void obterCliente() {}
        public void definirCliente() {}
        public void PartiparEvento() {}
}
```

25.



Arquivo do Astah em anexo no e-mail para melhor visualização.

Vantagem: Aumento encapsulamento e queda do acoplamento

Desvantagem: Queda no desempenho

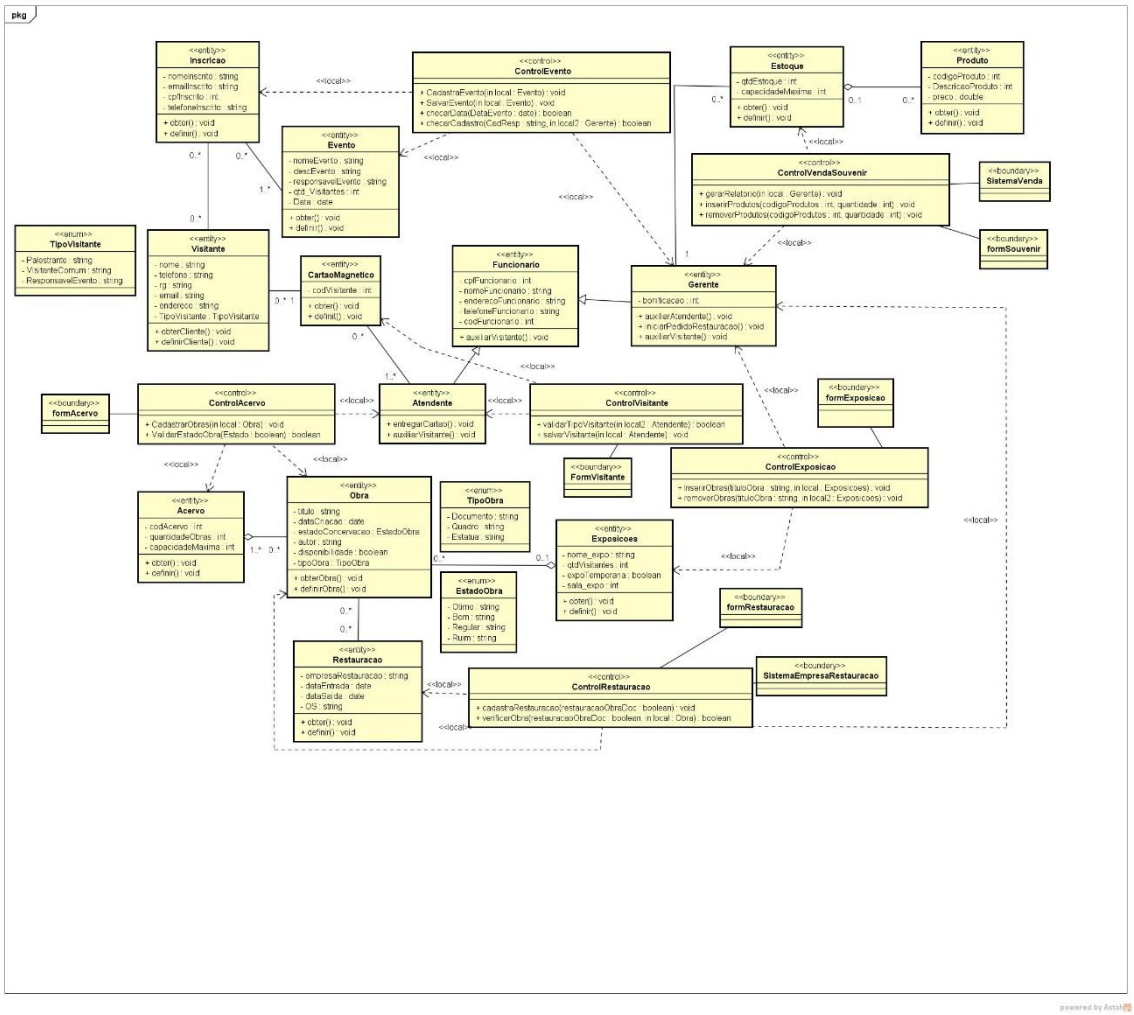
26.

```
public class ControlAcervo {

	private formAcervo formAcervo;
	public void CadastrarObras() {}
	public boolean ValidarEstadoObra(boolean Estado) {
		return false;
	}
}
public class ControlEvento {
```

```
        public void CadastraEvento() {}
        public void SalvarEvento() {}
        public boolean checarData(date DataEvento) {
                return false;
        }
        public boolean checarCadastro(string CadResp) {
                return false;
        }
}
public class ControlExposicao {

        private formExposicao formExposicao;
        public void InserirObras(string tituloObra) {}
        public void removerObras(string tituloObra) {}
}
public class ControlRestauracao {

        private formRestauracao formRestauracao;
        private SistemaEmpresaRestauracao sistemaEmpresaRestauracao;
        public void cadastraRestauracao(boolean restauracaoObraDoc) {}
        public boolean verificarObra(boolean restauracaoObraDoc) {
                return false;
        }
}
public class ControlVendaSouvenir {

        private SistemaVenda sistemaVenda;
        private formSouvenir formSouvenir;
        public void gerarRelatorio() {}
```

```
        public void inserirProdutos(int codigoProdutos, int quantidade) {}
        public void removerProdutos(int codigoProdutos, int quantidade) {}
}
public class ControlVisitante {

        private FormVisitante formVisitante;
        public boolean validarTipoVisitante() {
                return false;
        }
        public void salvarVisitante() {       }
}
```

27.



Arquivo do Astah em anexo no e-mail para melhor visualização.

Vantagens: Maior será o encapsulamento, por ser uma variável local estará mais protegida e só o método respectivo conseguirá visualizar, pois ela estará no código.

Desvantagens: Desempenho menor, pois a classe deverá ir buscar as informações, pois ela não está fixa em um método.

28.

```
public class controlVisitante{

        public boolean validarTipoVisitante(in local2 : Atendente){

        }
        public void salvarVisitante(in local : Atendente){

        }
}

public class controlEvento{

        public void CadastraEvento(in local : Evento){

        }
        public void SalvarEvento(in local : Evento) {
```

```
        }
        public boolean checarCadastro(CadResp : string, in local2 : Gerente){
        }
}


public class controlAcervo{

        public void CadastrarObras(in local : Obra){
        }
        public boolean ValidarEstadoObra(Estado : boolean) {
        }
}


public class controlrestauracao{

        public void cadastraRestauracao(restauracaoObraDoc : boolean){
        }
        public boolean verificarObra(restauracaoObraDoc : boolean, in local :
Obra) {
        }
}

public class controlExposicao{


        public void InserirObras(tituloObra : string, in local : Exposicoes) {
        }
        public void removerObras(tituloObra : string, in local2 : Exposicoes) {
```

```
        }
}
```

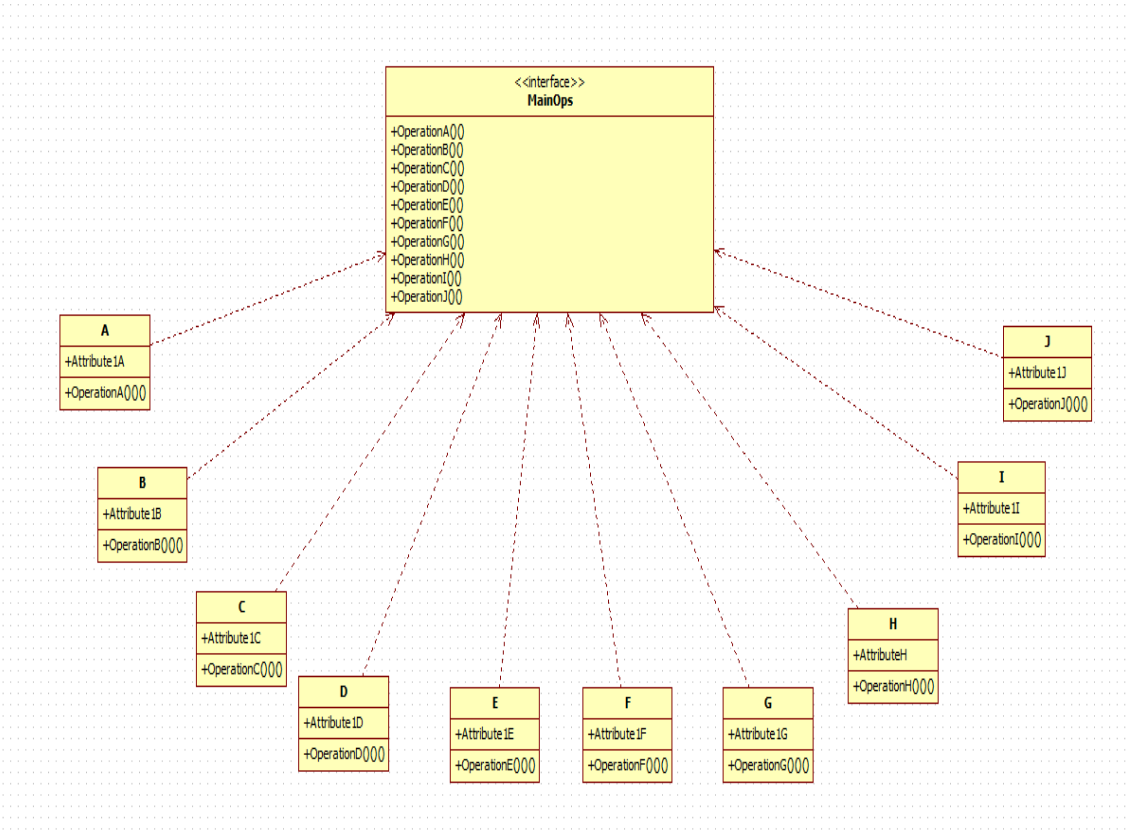
```
public class controlVendaSouvenier{

        public void gerarRelatorio(in local : Gerente){
                }
        Public void inserirProdutos(codigoProdutos : int, quantidade : int) {
                }
        Public void removerProdutos(codigoProdutos : int, quantidade : int) {
                }
}
```

Parte B

No diagrama apresentado, é possível notar forte acoplamento entre as classes (cada classe possui muitas dependências em relação a outras), porém fraca coesão, isto é, cada classe possui muitas responsabilidades e interações (dependências) entre si. Estas características podem gerar erros como: dificuldade de manutenção (uma correção em uma parte do sistema pode gerar novos erros); reutilização dos componentes do software, uma vez que devido à alta interdependência das classes torna-se difícil separar um componente sem que não dependa de mais classes.

No novo diagrama apresentado foi realizado a melhor coesão nas classes, atribuindo os atributos e métodos a respectiva classe, de modo que cada classe realizasse apenas a sua função específica.

Em relação ao acoplamento o novo modelo centraliza os métodos em uma interface, de tal forma a reduzir a dependência entre as classes.


<<interface>>
MainOps
+OperationA()()
+OperationB()()
+OperationC()()
+OperationD()()
+OperationE()()
+OperationF()()
+OperationG()()
+OperationH()()
+OperationI()()
+OperationJ()()
A
+Attribute1A
+OperationA()()()
B
+Attribute1B
+OperationB()()()
C
+Attribute1C
+OperationC()()()
D
+Attribute1D
+OperationD()()()
E
+Attribute1E
+OperationE()()()
F
+Attribute1F
+OperationF()()()
G
+Attribute1G
+OperationG()()()
H
+AttributeH
+OperationH()()()
I
+Attribute1I
+OperationI()()()
J
+Attribute1J
+OperationJ()()()