UFRPE UAST Unidade Acadêmica de Serra Talhada - PE

BSi Bacharelado em Sistemas de Informação

MPOO

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery **Data:** 16 / 09 / 2024

Aluno: _____

Nota:_____

2ª VERIFICAÇÃO DE APRENDIZAGEM

Instruções gerais:

- Utilize um diretório para salvar as implementações. Salve as implementações a cada modificação, caso aconteça alguma falha de energia o trabalho será preservado. Lembre-se que uma vez removido o arquivo do Eclipse, seu conteúdo será perdido.
- *A Nota máxima desta prova é de 10,0 pontos*. A Pontuação da Prova está distribuída entre os conceitos vistos na disciplina MPOO CCMP5012: (i) Projeto Eclipse e organização, (ii) Manipulação de Exceção, (iii) Composição, (iv) Thread, (v) Componentes gráficos, (vi) Eventos e tratamentos e (vii) Design Pattern.

---

1) No Eclipse limpe todos os projetos existentes.
   - Importe o **projeto** disponibilizado no SIGAA:
     br.2va.mpoo.edu.NomeSobrenome
     Este possui **uma pasta de pacotes chamada mpooshopsystem** que deverá conter todos os arquivos necessários para as respectivas questões.
   - Ao finalizar a prova **compacte o projeto** contendo toda a codificação do projeto (arquivos texto, *bytecodes e* imagens) e envio-o no SIGAA:
     [2ª Verificação de Aprendizagem] em Semana 14
     O SIGAA aceitará submissões até às 22h.

2) O proprietário de "MPOO Shop" precisa de um sistema para cadastrar clientes e distribuir cupons de descontos e disponibilizá-lo em um totem recém-adquirido. O sistema é descrito nas questões abaixo, modelado no APÊNDICE A e deverá ser implemente em Java.

***Design* Pattern (0,5 ponto)**

a) Utilize o padrão de arquitetura de projeto Model-View-Controller (MVC).

Agregação e Manipulação da base (1,0 ponto)

b) Implemente as definições do modelo do diagrama de classes do Apêndice A. Algumas definições do modelo do diagrama de classes do Apêndice A já são disponibilizadas. Portanto, acesse o AVA e baixe a codificação. É descrição:
   - Respeite as definições do diagrama.
   - Defina getters e setters apenas quando necessário;
   - BaseDados é uma classe que contém uma estrutura de dados para clientes do sistema. **APENAS** devem ser implementados os métodos apresentados no diagrama. Toda lógica necessária deve estar presentes exclusivamente nesses métodos;
   - A codificação deve aproveitar comportamentos já definidos, evitando a duplicidade de programação;
   - Há apenas uma base no sistema;
   - Um cliente é identificado pelo seu CPF e por seu e-mail;
   - Um cliente só poderá ser cadastrado uma única vez;
   - Os clientes cadastrados podem ganhar apenas um cupom de desconto; mas se não cadastrado este poderá ser cadastrado para ganhar um cupom;
   - O valor do cupom é de R$ 50,00 e só há três cupons disponíveis.
   - O cupom poderá ser resgatado enquanto houver disponíveis ou enquanto durar a promoção.
   - Textos personalizados devem utilizar a classe MensagemView. Faça o devido uso dos avisos nas situações do sistema.

Composição (1,0 ponto)

c) É relação de composição entre:
   - Cliente e Cupom
   - Cliente e Telefone

Métodos e Manipulação de Exceção (1,5 pontos)

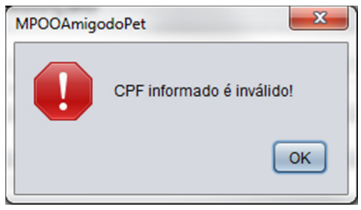
d) Faça a devida manipulação de Exception.

Em chamadas de métodos deve-se utilizar `try-catch`; e

Utilize as cláusulas `throws-throw` nas definições das exceções.

- Um cliente poderá ter o CPF validado pelo método `validarCPF (String CPF)` da classe `ValidadorCPF`. Se inválido, um cliente não poderá fazer parte do sistema. O método de validação está presente na codificação disponibilizada, entretanto se deve modifica-la de maneira a levantar `CPFException`.

Personalize a mensagem para notificar o usuário do sistema quando essa exceção ocorrer:



`exibirMensagemErro(mensagem : String)`

- Caso se tente cadastrar um cliente já cadastrado deve-se levantar a exceção `CadastroException` no método `adicionarCliente` de BaseDados.
- No médodo `adicionarCupom` de `BaseDados`:
  - Ao tentar adicionar um cupom a um cliente não cadastrado deve-se levantar a exceção ClienteException.
  - Também se tentar adicionar um cupom quando os disponíveis estiverem esgotados deve-se levantar CupomEsgotadoException.

Relançamento de Exceção e seu Tratamento (0,5 ponto)

A base ao ser inicializada deve adicionar os clientes:
- Nome: `"José Alves"`, cpf: `"057.267.536-40"`, email: `"jose@gmail.com"`, e fone: `"55(81)99999-0000"`; e
- Nome: `"João Santos"`, cpf: `"013.671.546-00"`, email: `"joao@gmail.com"`, e fone: `"55(87)99999-1111"`.

Reflita o Relançamento de Exceção para a não violar o MVC, de maneira que o model não exiba notificação de erro com apresentação de view, consequentemente, a responsabilidade é passada aos pacotes app e/ou controller.

Thread (1,0 ponto)

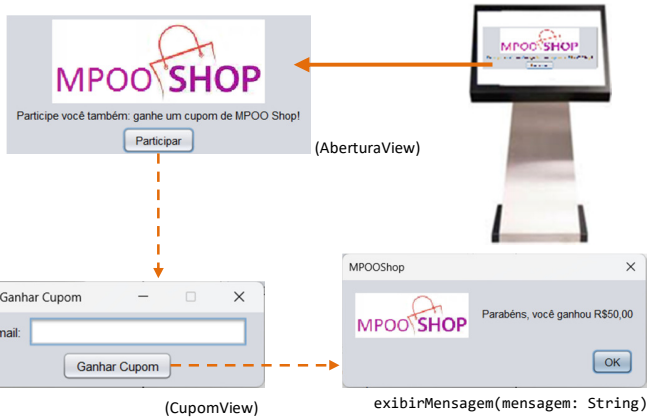
e) O sistema possui o robô GerenciadorCupom que mantém ativa a promoção de distribuição de cupons. Depois de passados "10 segundos" o sistema exibe a mensagem de encerramento e finaliza o sistema:

`exibirFimPromocao()`

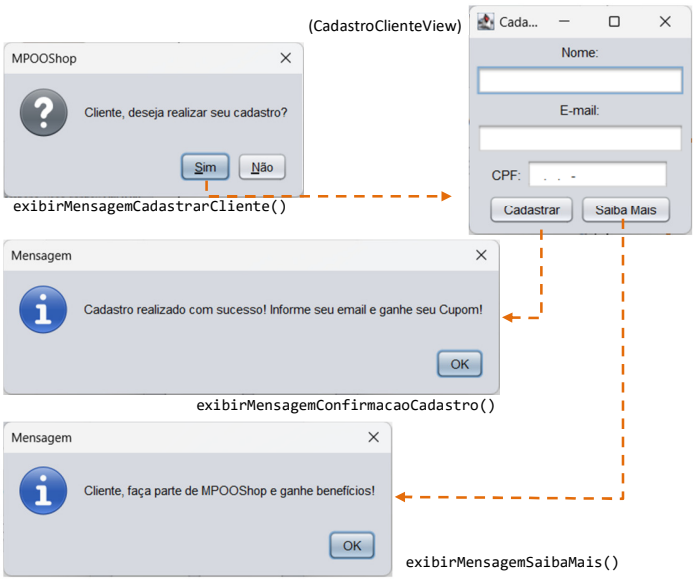
Componentes Gráficos (2,0 pontos)

É descrição das GUI's do sistema MPOOShop:

f) Possui uma tela a ser exibida no Totem para interação do usuário:



Se um cliente não cadastrado tentar acessar o sistema, deve-se exibir:



Há apenas uma janela ativa e uma única instância ativa para cada tela. Apenas AberturaView se fechada encerra o sistema. Logo, tela CadastroClienteView não encerra o sistema, logo se fechada o sistema deverá voltar a exibir CupomView (esta última se fechada deverá voltar a AberturaView).

Utilize Look and feel:

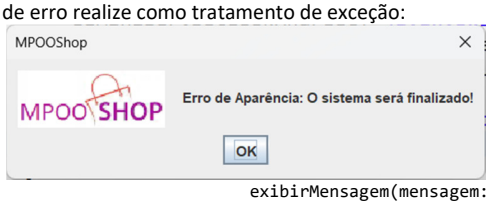
```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel")
```

Métodos e Manipulação de Exceção (0,25 pontos)

Utilize Look and feel:

```
UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel")
```

Em caso de erro realize como tratamento de exceção:



exibirMensagem(mensagem: String)

Utilize o gerenciador de layout FlowLayout.

São componentes:

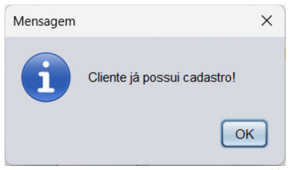
`JFrame, JLabel, JTextField, JFormattedTextField, JButton e JOptionPane`

Dica:

```
JFormattedTextField cpfField;
try {
     cpfField = new JFormattedTextField(
                                  new MaskFormatter("###.###.###-##")
                                        );
     cpfField.setColumns(10);
   } catch (ParseException e) {}
```

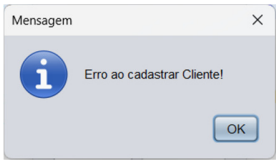
Pode-se ainda ter como mensagem:

- Cliente já possui cadastro:



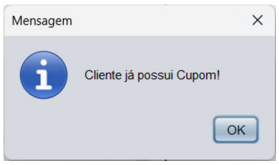
Usar `exception.getMessage()`

- Erro ao cadastrar cliente:



exibirMensagemErroCadastroCliente()

- Erro quando cliente já possui cupom e tenta receber outro:



exibirMensagemErroCupomCliente()

Eventos e Tratamentos (2,5 pontos)
(Bônus) Design Pattern: Adapter (0,25 ponto)

g) Os tratamentos dos eventos das telas (view) devem estar nos respectivos controladores (controller) conforme diagrama de classes do Apêndice A:

- Para IndexView tem-se IndexController, sendo:
  - Tratamento de evento por classe interna privada:
    - ButtonHandler: Quando o cliente aciona o botão Adotar;
    - KeyHandler: Para tratar o encerramento do sistema por ESC; e Para acionar o botão por ENTER.
- Para CadastroClienteView tem-se CadastroClienteController, sendo:
  - Tratamento de evento realizado por método sobrescrito por classe (classe realiza a interface):
- Para CupomView tem-se CupomController, sendo:
  - Tratamento de evento realizado por método em classe ointerna anônima

Eventos e Tratamentos (0,5 ponto)
(Bônus) Design Pattern: Adapter (0,25 ponto)

Como as telas CadastroClienteView e CupomView não encerram o sistema e é preciso retornar a outra tela, então:

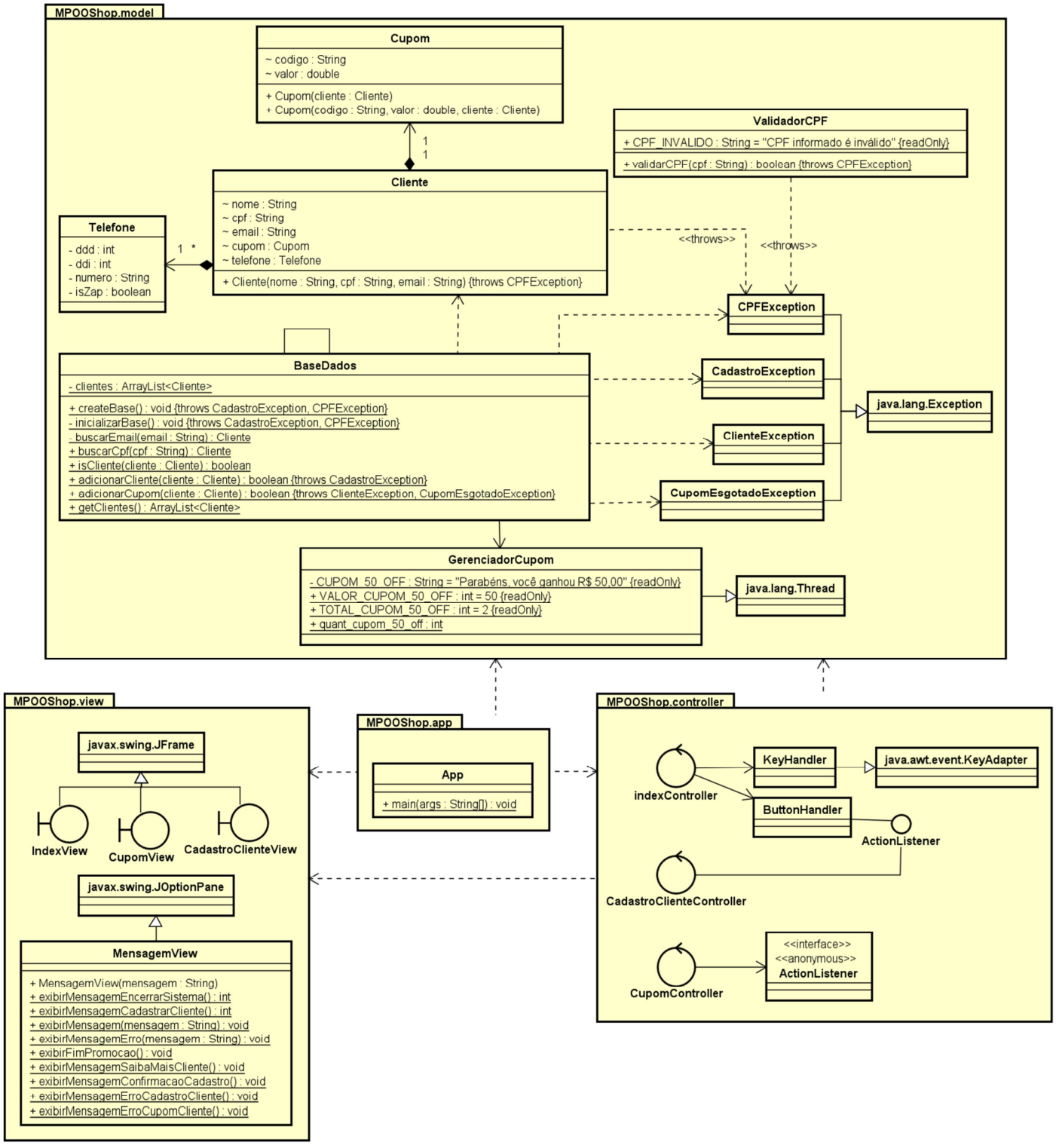
- Tratamento de evento por classe interna privada:
  - WindowHandler: Para saber se a tela foi fechada.

Manipulação das definições

h) Deve-se ter em App:

- A criação do robô (Threads)
- A criação das instâncias MVC

APÊNDICE A


MPOOShop.model
Cupom
~ codigo : String
~ valor : double
+ Cupom(cliente : Cliente)
+ Cupom(codigo : String, valor : double, cliente : Cliente)
1
1
ValidadorCPF
+ CPF_INVALIDO : String = "CPF informado é inválido" {readOnly}
+ validarCPF(cpf : String) : boolean {throws CPFException}
Cliente
~ nome : String
~ cpf : String
~ email : String
~ cupom : Cupom
~ telefone : Telefone
+ Cliente(nome : String, cpf : String, email : String) {throws CPFException}
Telefone
- ddd : int
- ddi : int
- numero : String
- isZap : boolean
1 *
<<throws>>
<<throws>>
CPFException
BaseDados
- clientes : ArrayList<Cliente>
+ createBase() : void {throws CadastroException, CPFException}
- inicializarBase() : void {throws CadastroException, CPFException}
- buscarEmail(email : String) : Cliente
+ buscarCpf(cpf : String) : Cliente
+ isCliente(cliente : Cliente) : boolean
+ adicionarCliente(cliente : Cliente) : boolean {throws CadastroException}
+ adicionarCupom(cliente : Cliente) : boolean {throws ClienteException, CupomEsgotadoException}
+ getClientes() : ArrayList<Cliente>
CadastroException
ClienteException
CupomEsgotadoException
java.lang.Exception
GerenciadorCupom
- CUPOM_50_OFF : String = "Parabéns, você ganhou R$ 50,00" {readOnly}
+ VALOR_CUPOM_50_OFF : int = 50 {readOnly}
+ TOTAL_CUPOM_50_OFF : int = 2 {readOnly}
+ quant_cupom_50_off : int
java.lang.Thread
MPOOShop.view
javax.swing.JFrame
IndexView
CupomView
CadastroClienteView
javax.swing.JOptionPane
MensagemView
+ MensagemView(mensagem : String)
+ exibirMensagemEncerrarSistema() : int
+ exibirMensagemCadastrarCliente() : int
+ exibirMensagem(mensagem : String) : void
+ exibirMensagemErro(mensagem : String) : void
+ exibirFimPromocao() : void
+ exibirMensagemSaibaMaisCliente() : void
+ exibirMensagemConfirmacaoCadastro() : void
+ exibirMensagemErroCadastroCliente() : void
+ exibirMensagemErroCupomCliente() : void
MPOOShop.app
App
+ main(args : String[]) : void
MPOOShop.controller
indexController
KeyHandler
java.awt.event.KeyAdapter
ButtonHandler
ActionListener
CadastroClienteController
CupomController
<<interface>>
<<anonymous>>
ActionListener