

3ª VERIFICAÇÃO DE APRENDIZAGEM

Instruções gerais:

- No Eclipse crie um novo projeto chamado **br.com.mpoo.3va.NomeSobrenome**, que deverá ter a pasta de pacotes "mpooSystem" com todos os pacotes e classes das questões.
- 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 contendo será perdido.
- A prova é prática e deverá ser entregue no AVA em [3 V.A. Prova Teórico-Prática] em Semana 15 da seção Atividades. Deverá ser entregue todo o projeto Eclipse contendo os códigos-fonte implementados em Java. A Nota máxima desta prova é de 10,0 pontos, com pontuação distribuída entre os conceitos vistos na disciplina MPOO CCMP5012.

Criação e Configuração de Projeto:

1) No Eclipse:

- Crie um novo projeto chamado **br.com.mpoo.3va.NomeSobrenome**
Este deverá ter uma pasta de pacotes chamada **mpooSystem** contendo todos os arquivos necessários para as próximas questões.
- Ao finalizar a prova **compacte o projeto** contendo toda a codificação do projeto (arquivos texto (.java), *bytecodes (.class)* e imagens) e envie-o no AVA :
 - [3 V.A. - Prova Teórico-Prática] em Semana 15 da seção Atividades
 - O AVA aceitará submissões **conforme regras de envio**, na qual deverá ser entregue partes do sistema conforme especificado.

2) O sistema descrito abaixo é modelado no APÊNDICE A e deverá ser implemente em Java.

Regra de envio: até 30 minutos após o início da prova

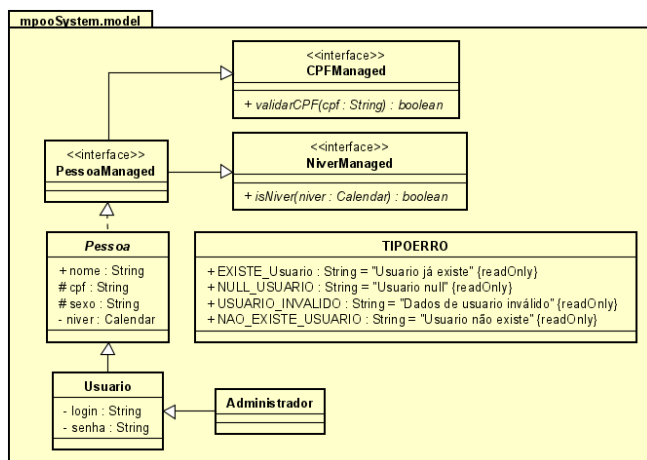
Design Pattern (1,0 ponto)

a) Utilize o padrão de projeto: Model-View-Controller (MVC). São pacotes: model, view, controller, util e app.

Classe, atributo, Método (construtor e concreto) e Encapsulamento (0,5 ponto)
Herança (0,5 ponto)
Interface (0,5 ponto)

b) Implemente:

- O diagrama abaixo:



Obs.: A interface CPFManaged que obriga a verificação de um cpf :

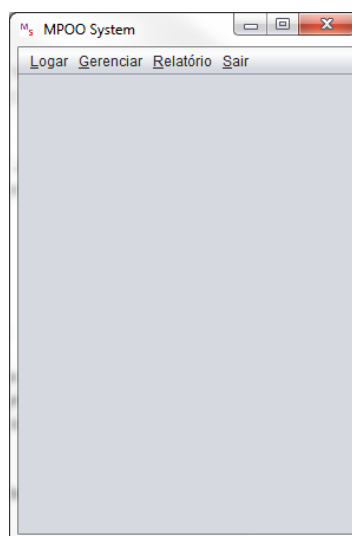
```
public boolean validarCPF(String CPF){}
```

Tem sua implementação disponível no AVA.

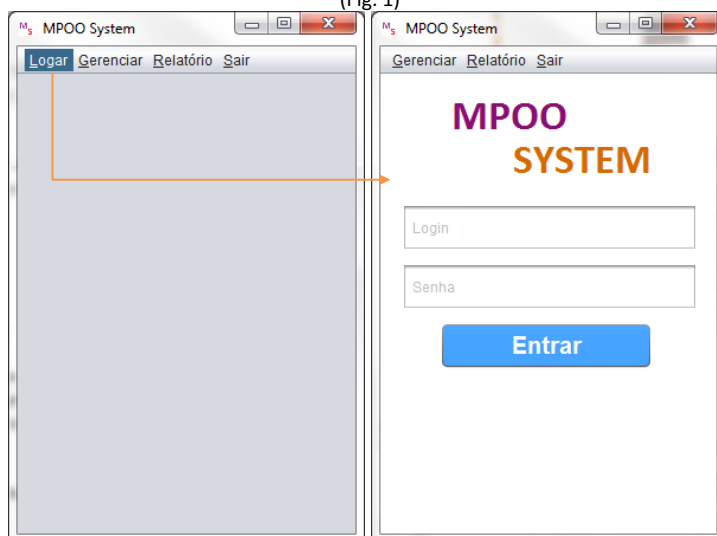
Regra de envio: até 2h após o início da prova

Componentes gráficos (1,5 pontos)

- Todas as mensagens devem utilizar a classe MensagemGUI, vide Apêndice A.
- Ao executar a App tem-se a janela personalizada SistemaGUI, vide Apêndice A.



(Fig. 1)



(Fig. 2A)

(Fig. 2B)



(Fig. 3A)

(Fig. 3B)

As telas possuem as seguintes descrições:

• **Look and feel:**

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

- Componentes: JFrame, JPanel, JMenuBar/JMenuItem, JLabel, JTextField, JButton, JRadioButton e JPasswordField.

• Layouts:

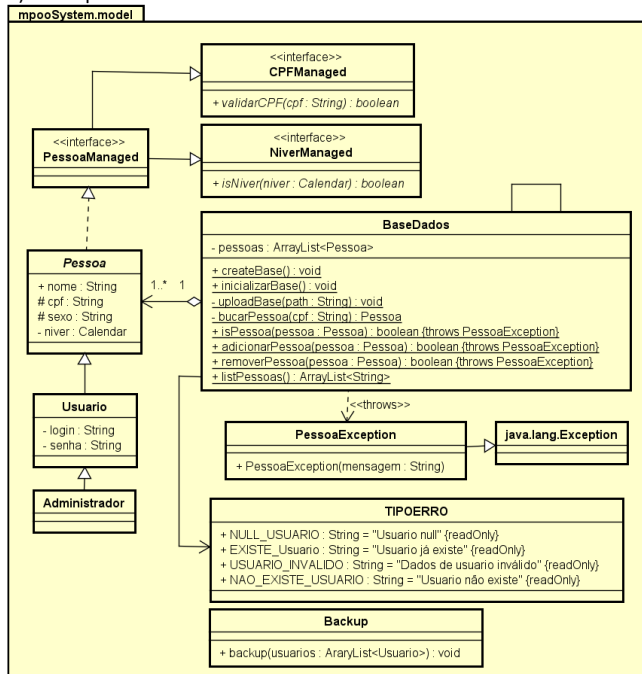
- null, FlowLayout e SpringLayout.

- Só há um JFrame, os campos da Fig. 2B e Fig. 3B utilizam JPanel.

Após duas 2 horas de provas

Regra de envio: até 4 horas após o início da prova

e) Complete o model do sistema:



Agregação, Dependências e Relações, ArrayList, Métodos e utilização (1,0 ponto)

É descrição da BaseDados:

- BaseDados é uma classe que contém uma estrutura de dado para as pessoas do sistema. APENAS devem ser implementados os métodos apresentados no diagrama. Toda lógica necessária deve estar presentes exclusivamente nesses métodos.
- Há apenas uma base no sistema;

- Faça uso de polimorfismo de objetos.
- cpf é chave primária de uma pessoa;
- Não deverá ter duplicidade de código, portanto, deve-se fazer uso de métodos existentes e àqueles disponíveis em ArrayList;
- Textos personalizados ds mensagens de *feedback* ao usuário devem utilizar as constantes de TipoErro.
- Apenas usuários do tipo Administrador podem realizar a manutenção da base via GUI;
- `listPessoas()` retorna o nome de todos os usuários da base.

Métodos, tratamento e Manipulação de Exceção (1,0 ponto)

f) Impedimento as regras de negócio para o tratamento de exceções. A partir do diagrama de classes do Apêndice A, observe a existência da classe PessoaException. Também deverão ser utilizadas as classes TipoErro e TelaMensagem. São regras:

- Os métodos `isPessoa`, `adicionarPessoa` e `removerPessoa` devem tratar a exceção `PessoaException` (**throws/throw**) quando uma pessoa for **null**, se já pertencer a base (`isPessoa(pessoa)==true`) em uma adição ou se uma pessoa não pertencer a base (`isPessoa(pessoa)==false`) em uma remoção. Caso contrário deverá ter-se o retorno booleano dos métodos de ArrayList para adição/remoção.

g) Faça o devido uso de **try/catch** na chamada desses métodos.

Manipulação de Arquivo (XML: 1,0 ponto; TXT: 0,5 ponto)

h) O método `inicializarBase()` de `BaseDados` deve carregar o usuário abaixo assim que o sistema for inicializado:

```

<?xml version='1.0' encoding='UTF-8'?>
<Usuarios>
  <Usuario>
    <nome>José Silva</nome>
    <cpf> 80436150395</cpf>
    <sexo>masculino</sex>
    <niver>
      <time>1578006019610</time>
      <timezone>GMT-03:00</timezone>
    </niver>
    <login>zesilva</login>
    <senha>zeSilvA123</senha>
  </Usuario>
</Usuarios>
  
```

i) Os dados são salvos em XML. Utilize o método da classe Backup: `backup (usuarios: ArrayList<Usuario>)`

Esse método deverá ser utilizado sempre que um usuário seja adicionado a base ou pela thread do sistema

Thread (0,5 ponto)

j) App é uma Thread que a cada 10 minutos exibe os nomes dos usuários cadastrados em um JOptionPane (proponha um, mas respeitando MVC) que estão fazendo aniversário. Para isso utilize:

- O método `listPessoas()` de `BaseDados`;
- O método `isNiver` da interface `NiverManaged`:

```

public boolean isNiver(Calendar niver){
    if ((niver.get(Calendar.MONTH)+1)==LocalDateTime.now().getMonthValue())
        return true;
    else return false;
}
  
```


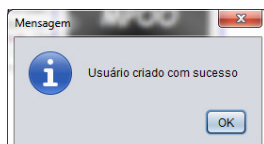
Eventos e tratamentos: 1,5 pontos
(Bônus: Adapter: 0,25 pontos)

k) São funcionalidades e tratamentos:

- Todos os **menus** devem ser tratados na própria classe do controlador, assumindo o fato de **o controlador realizar interface**:
 - Logar – Libera os campos para login no sistema.
 - Gerenciar -> Usuario -> Adicionar– Libera os campos para cadastrar um usuário no sistema, mas apenas se um administrador estiver logado no sistema, caso contrário, deve exibir os campos para login.
 - Relatório – Exibe todos os nomes dos usuários cadastrados no sistema. Proponha um JPanel que deve ser exibido no JFrame do sistema
 - Sair – Encerra a aplicação.

- Uso adequado de conceitos de OO (0.5 ponto)

Se a criação for realizada com sucesso exibe-se a mensagem:



Usuário já existe

OK

(Bônus) Eventos e tratamentos (1.0 ponto)

n) Deve-se ter em App:

- A criação de instâncias MVC
- A criação de um administrador para a base: nome: “seu nome”, cpf: “seu cpf”, sexo: “seu sexo”, niver: “seu niver”, login: admin e senha: admin;
- A criação do usuário: nome: João Silva, cpf: 584.084.062-99, sexo: masculino, niver: 23/05/1980, login: joaoSilva e senha: joaoSilva;
- Ilustre a inserção de um usuário com cpf inválido.
- Tratamentos por **try/catch** nas chamadas de métodos.
- Inicialização da Thread do sistema.

Manipulação das definições (1,5 pontos)

```
public class MP00 {
    void saudacao(){
        System.out.println("Boa Prova!");
    }
    public static void main(String[] args) {
        new MP00().saudacao();
    }
}
```

The UML diagrams for the **mpooSystem** project are organized into four main packages:

- mpooSystem.model**: Contains the domain logic.
 - Interfaces**: `PessoaManaged` (with `CPFManaged` sub-interface) and `NiverManaged`.
 - BaseDados**: A class managing a list of `Pessoa` objects, with methods like `createBase()`, `inicializarBase()`, `uploadBase()`, `bucarPessoa()`, `isPessoa()`, `adicionarPessoa()`, `removerPessoa()`, and `listPessoas()`.
 - Pessoa**: A class implementing `PessoaManaged` and `NiverManaged`, with attributes `nome`, `cpf`, `sexo`, and `niver`.
 - Usuario**: A class with `login` and `senha` attributes.
 - Administrador**: A class extending `Usuario`.
 - Exceptions**: `PessoaException` (extending `java.lang.Exception`) and `TipoErro` (with constants for various error types).
 - Backup**: A class with a `backup(usuarios)` method.
- mpooSystem.view**: Contains the user interface components.
 - SistemaGUI**: A class that manages `LoginPanel` and `CadastroPanel`.
 - MensagemGUI**: A class for displaying messages, with a `exibirMensagem()` method.
 - javafx.swing.JFrame** and **javafx.swing.JOptionPane**: Standard Java Swing components.
- mpooSystem.controller**: Contains the logic for handling user input.
 - MpooSystemController**: A class implementing `ActionListener` and `FormularioHandler`, with `FocusListener` and `CaretListener` sub-interfaces.
- mpooSystem.util**: Contains utility classes.
 - SpringUtilities**: A utility class for Spring framework integration.
- mpooSystem.app**: Contains the application entry point.
 - App**: A class with a `main(args)` method, which starts the application.
 - Thread**: A class representing the application's execution thread.

The diagrams illustrate the relationships between these components, including inheritance, associations, and dependencies.