INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

Prova 3 – Programação Orientada a Objetos – PROO
Prof. Lucas Oliveira

PRONTUÁRIO: INDICAR NO ZIP Dia: 27/06/2024 Horário: 08h00 – 11h30

✓ A prova é individual e sem consulta. ✓ Não é permitido utilizar qualquer código implementado por você anteriormente. ✓ Não coloque seu nome no projeto. ✓ Atribui-se nota zero à prova em desacordo com os itens acima.	✓ A prova deve ser nomeada da seguinte forma: PRONTUARIO_P3, com o "SC". ✓ Não envie apenas as classes, mas todo o projeto. ✓ Envie o projeto como zip no Moodle.

Descrição do contexto – Você continua trabalhando no seu sistema para cálculo de comissões de vendas. Agora que as regras de negócio já foram criadas nas classes do domínio, é necessário implementar a persistência e desenvolver a interface gráfica. Seu sócio na startup já implementou uma classe chamada *DatabaseBuilder* (pacote *persistence*), que cria o banco de dados no SQLite. Ele também já implementou a interface gráfica principal, chamada *MainView* (pacote *view*), que exibe os dados de todos os funcionários da empresa. Como vocês viram na disciplina de POO, é uma boa prática separar regras de negócio das regras de persistência. Por isso, além de adotar uma interface *EmployeeDAO*, agora vocês trabalham apenas com Objetos de Transferência de Dados (DTOs) na camada de persistência. Todas as regras de negócio sobre processos da aplicação devem ser implementadas exclusivamente em classes de serviço. Neste contexto, você deve trabalhar para concluir aplicação, fazendo a transição de um banco de dados falso, na memória, para um banco de dados relacional, além de oferecer as operações CRUD a partir de interfaces gráficas.

Relembrando o domínio do problema – Algumas empresas com Natura e HerbaLife possuem um sistema de vendas que funciona por meio de consultores. Um Revendedor (*Reseller*), ao atingir um grande número em vendas, pode empregar outros funcionários para que eles o auxiliem, tornando-se um Consultor (*Consultant*). Um Funcionário (*Employee*) pode ser tanto um simples revendedor quanto um consultor. Revendedores ganham 15% do valor bruto de suas vendas. Consultores ganham, além das porcentagens de suas vendas, o equivalente a 30% do valor total das comissões recebidas pelos funcionários sob sua imediata responsabilidade. O cálculo do valor dos ganhos obtidos por consultores e revendedores de uma empresa demanda muito tempo e muitas vezes resulta em erros.

Execute as atividades a seguir para a implementação em Java. Para a atribuição da nota será levada em conta não apenas a funcionalidade, mas a qualidade, adequação e pertinência de cada solução. Bom senso faz parte da prova.

Dica 1: Há uma classe chamada *SandBox*, que pode ser utilizada para testes das implementações em andamento. Nada escrito nessa classe será corrigido, pois será considerado como rascunho.
Dica 2: Explore as classes existentes nos pacotes do projeto para verificar as funcionalidades já disponíveis.

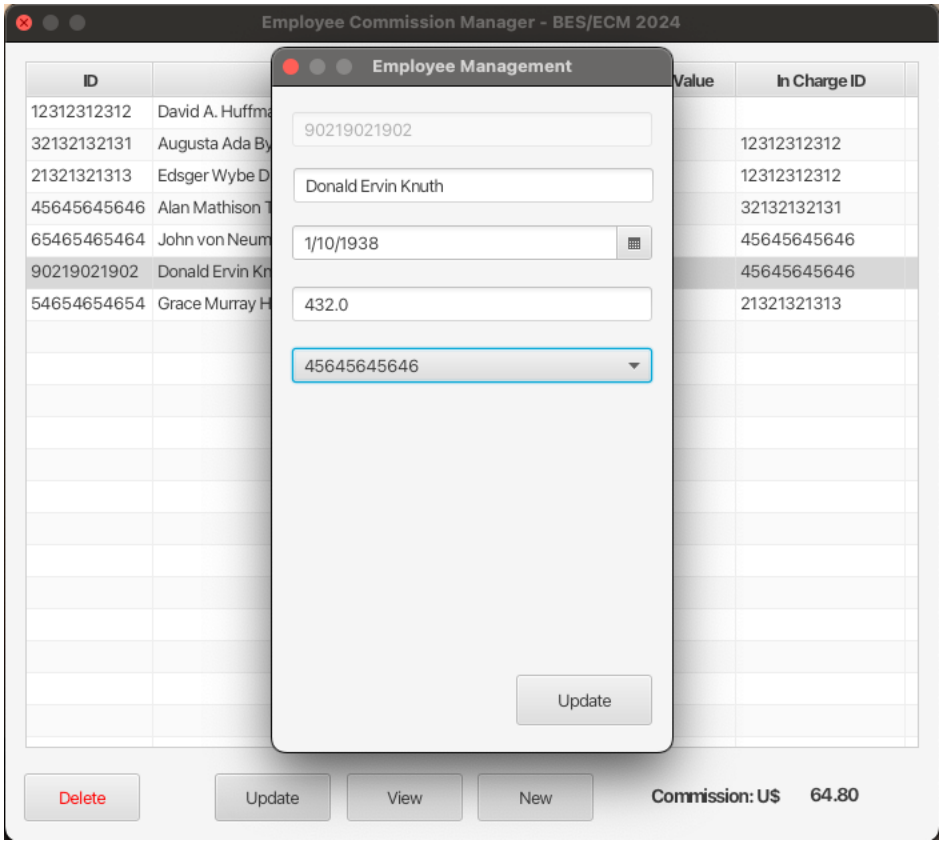
#	Descrição	Pont.
1	Crie uma classe chamada *RegisterEmployeeService* no pacote *services*, que receba um *EmployeeDAO* por injeção de dependência. Em um método *register()*, receba um *EmployeeDTO* para que ele seja persistido no banco de dados. Caso o registro já exista no banco de dados, lance uma *EntityAlreadyExistsException*. Se o responsável pelo novo funcionário for não nulo, mas não existir no banco de dados, lance uma *NoSuchElementException*. Não deve ser possível inserir um usuário nulo ou com id nulo no banco de dados.	0,5pt
2	Crie uma classe chamada *UpdateEmployeeService* no pacote *services*, que receba um *EmployeeDAO* por injeção de dependência. Em um método *update(),* receba um *EmployeeDTO* para que ele seja atualizado no banco de dados. Caso o registro não exista	0,5pt

	no banco de dados, lance uma *NoSuchElementException*. Se o responsável pelo funcionário sendo atualizado for não nulo, mas não existir no banco de dados, lance uma *NoSuchElementException*.	
3	Crie uma classe chamada *RemoveEmployeeService* no pacote *services*, que receba um *EmployeeDAO* por injeção de dependência. Em um método *remove()*, receba um id de funcionário para que ele seja removido do banco de dados. Caso o registro não exista no banco de dados, lance uma *NoSuchElementException*. Se o funcionário for um consultor, atualize todos os seus subordinados imediatos, para que o responsável deles seja o mesmo responsável do funcionário removido.	0,75pt
4	Implemente uma classe *ConnectionFactory* no pacote *persistence*, que crie sob demanda um objeto *Connection* único e que também prepare statements para atualizações e consultas no banco de dados.	0,75pt
5	Crie uma classe *SQLiteEmployeeDAO* que implemente a interface EmployeeDAO. Implemente o método *save()* para persistir funcionários no banco de dados. Dica: "INSERT INTO <table> (<columns>) VALUES (<values>))"	0,75pt
6	Na classe *SQLiteEmployeeDAO*, implemente os métodos *findOne()* e *findAll()* para obter, respectivamente, um funcionário por id e todos os funcionários do banco de dados. Dica 1: "SELECT <values> FROM <table> WHERE <criteria>". Dica 2: Para converter String em um LocalDate, utilize o método estático LocalDate.parse(String).	0,75pt
7	Na classe *SQLiteEmployeeDAO*, implemente o método *update()* para atualizar funcionários no banco de dados. Dica: "UPDATE <table> SET <col1> = <value1>, <colN> = <valueN> WHERE <criteria>"	0,5pt
8	Na classe *SQLiteEmployeeDAO*, implemente o método *delete()* para remover funcionários do banco de dados. Dica: "DELETE FROM <table> WHERE <criteria>"	0,5pt
9	Crie uma classe chamada *CommissionService* no pacote *services*, que receba um *EmployeeDAO* por injeção de dependência. Em um método *calculateCommission*(), receba um id de funcionário para que seja calculada sua comissão. Note que o cálculo da comissão é uma regra de negócio já implementada pelas classes do modelo (pacote *model*). Logo, para realizar o cálculo da comissão, você deve mapear o retorno dos registros do banco de dados (DTOs) de forma a compor objetos do modelo.	2,0pts
10	Crie uma interface gráfica para permitir a gestão de um funcionário. Configure sua abertura para modos de inclusão, atualização e remoção. Organize as classes e arquivos necessários em seus respectivos pacotes. Utilize como modelo a interface disponível no Apêndice A. Para tratar os componentes ComboBox e DatePicker, utilize os direcionamentos disponíveis no Apêndice B.	1,25pt
11	Integre todas as funcionalidades disponíveis na interface gráfica às regras de negócio desenvolvidas nas classes de serviço. Inclua no método tratador de evento *showCommission()* uma chamada para a funcionalidade de cálculo da comissão (Item 9), para que os valores sejam exibidos sempre que uma linha da tabela for selecionada.	1.75pt

O estudante que terminar todas as atividades corretamente primeiro ganha 1,0pt adicional para usar nas provas anteriores.

***** Boa sorte! ****

APÊNDICE A – Modelo de Interface Gráfica



APÊNDICE B – Implementação do ComboBox e do DatePicker

1. Para adicionar elementos em um ComboBox:

```
@FXML
public void initialize() {
    final List<String> inChargeEmployees = repository.findAll().stream().map(EmployeeDTO::id).toList();
    comboInCharge.setItems(FXCollections.observableList(inChargeEmployees));
}
```

2. Para obter um elemento do ComboBox:

```
comboInCharge.getSelectionModel().getSelectedItem()
```

3. Para selecionar um elemento do ComboBox:

```
comboInCharge.getSelectionModel().select(<VALOR_A_SER_SELECIONADO>);
```

4. Objetos da classe DatePicker recebem e selecionam datas a partir dos métodos getValue() e setValue(), que trabalham sempre com valores do tipo LocalDate.