**Sistema Bancário**

**Documento de Requisitos**

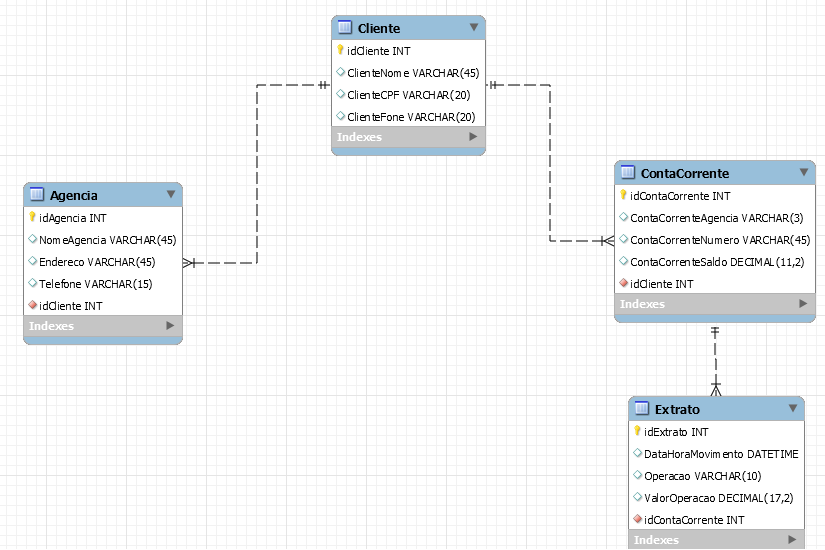
**Módulo Java Avançado**

**Versão 1.0.1**

O novo banco on-line BANK precisa desenvolver inicialmente um módulo essencial do sistema bancário, esse módulo consiste de uma API que irá armazenar as seguintes transações bancárias: Sacar, Depositar e Transferir numerário;

Abaixo está o modelo lógico do sistema bancário;

O modelo pode ser alterado/corrigido;



**Modelo ER**

* O módulo cadastro que contempla o armazenamento de dados do ***Conta corrente*** e ***Agência*** depende da *entidade* ***Cliente*** *que* deve armazenar os dados abaixo:

|  |  |
| --- | --- |
| **RF-01** | **ID** do cliente deve ser um número sequencial e inteiro de valor obrigatório gerado pelo banco;  \*Campo obrigatório; |
|  | **Nome** \*Campo obrigatório;  O sistema deve lançar uma exceção caso o campo esteja vazio; |
|  | **CPF** deve ser uma string de no máximo 14 caracteres;  O sistema deve lançar uma exceção de campo maior que 14 caracteres;  Validar CPF e lançar exceção caso o CPF não seja válido;  \*Campo obrigatório; |
|  | **Fone** O sistema deve lançar uma exceção caso o campo esteja vazio;  \*Campo obrigatório; |

O cadastro da ***agência*** deve armazenar os dados:

|  |  |
| --- | --- |
| **RF-02** | **IDAgencia** do aluno deve ser um número sequencial e inteiro de valor obrigatório gerado pelo banco;  \*Campo obrigatório; |
|  | **Nome**  O sistema deve lançar uma exceção caso o campo esteja vazio;  \*Campo obrigatório; |
|  | **Endereço**;  O sistema deve lançar uma exceção caso o campo esteja vazio;  \*Campo obrigatório; |
|  | **Telefone**  O sistema deve lançar uma exceção caso o campo esteja vazio;  \*Campo obrigatório; |
|  | **IDCliente** obtido da tabela de cliente;  \*Campo obrigatório; |

A entidade **Conta corrente** deve armazenar os dados:.

|  |  |
| --- | --- |
| **RF-03** | **IDContaCorrente** deve ser um número sequencial e inteiro de valor obrigatório gerado pelo banco;  \*Campo obrigatório; |
|  | **ContaCorrenteAgencia** Obtido da tabela de Agência  \*Campo obrigatório; |
|  | **ContaCorrenteNumero**  O sistema deve lançar uma exceção caso o campo esteja vazio;  \*Campo obrigatório; |
|  | **IDCliente** obtido da tabela de cliente  \*Campo obrigatório; |

A entidade **Extrato** deve armazenar os dados:

|  |  |
| --- | --- |
| **RF-04** | **ID** da comunicação deve ser um número sequencial e inteiro de valor obrigatório gerado pelo banco;  \*Campo obrigatório; |
|  | **dataHoraMovmento** deve guardar a data e hora que foi feita a movimentação;  \*Campo obrigatório; |
|  | **Operação** Enumeration que indica se a operação foi um saque, ou um depósito, ou transferencia;  \*Campo obrigatório; |
|  | **IDContaCorrente** obtido da tabela de conta corrente;  \*Campo obrigatório; |

Com base nas informações anteriores defina o mapeamento das classes de entidades (Entities) utilizando JPA, siga os seguintes passos:

1. Verifique que JDK versão 8 corretamente instalada no seu computador, caso contrário pode instalar desde o instalador que vem na pasta *ambiente* no material deste curso.
2. Verifique que a variável de ambiente JAVA\_HOME esteja definida corretamente e apontando para a pasta de instalação do JDK.
3. Execute o IDE Eclipse que vem na pasta *ambiente* no material deste curso.
4. Crie um pacote com.accenture.academico.model e defina as entidades e mapeamentos em JPA de acordo com a logica de tabelas e entidades definidas.

Utilize os mapeamentos nas relações utilizando anotattions

@OneToMany

@ManyToMany

@ManyToOne

@OneToOne

Segundo seja necessário, defina as relações de herança e composição e a definição dos campos descritos nas tabelas anteriores, respeitando as *constraints* e definição de campos.

1. Crie os Enums que achar convenientes segundo seja o caso e vincule-os nos campos das entidades
2. Defina os DAOs com operações CRUD básicas para acessar, modificar e remover os dados das entities definidas;
3. Execute os testes de cadastro, consulta utilizando classes de teste JUnit;
4. Crie o método para fazer depósito na conta corrente;
5. Crie o método para fazer saques na conta corrente;
6. Crie um método para consultar os dados da conta de uma cliente;
7. Crie o método para fazer transferência entre contas
8. Na transferência verifique se a conta destino existe, caso não exista lance uma exceção customizada;
9. Crie um método para exibir o extrato de uma conta;
10. Crie um método para recalcular o saldo de um cliente;