Se para cada criação de conta se deve consultar o serasa

```
	public Conta registraConta(Cliente titular, int numero) throws
SQLException {
		Calendar hoje = Calendar.getInstance();

		Conta novaConta = new Conta(numero, hoje);

		novaConta.setTitular(titular);

		Dao<Conta> dao = new Dao<Conta>();
		dao.adiciona(novaConta);

		return novaConta;
	}
```

O banco deve consultar o serasa antes de criar a conta

```
	public Conta registraConta(Cliente titular, int numero) throws
SQLException {

		if (consultaSerasa(titular.getCpf())) {

			Calendar hoje = Calendar.getInstance();

			Conta novaConta = new Conta(numero, hoje);

			novaConta.setTitular(titular);

			Dao<Conta> dao = new Dao<Conta>();
			dao.adiciona(novaConta);

			return novaConta;
		}

		return null;
	}
```

Essa classe ainda faz persistência e banco de dado e controle de regra de negocio, as funções devem ser divididas

```
	public Conta criaConta(Cliente titular, int numero) throws SQLException {

		if (consultaSerasa(titular.getCpf())) {

			Calendar hoje = Calendar.getInstance();
			Conta novaConta = new Conta(numero, hoje);

			novaConta.setTitular(titular);
			registraConta(novaConta);

			return novaConta;
		}

		return null;
	}

	public Boolean registraConta(Conta conta) throws SQLException {

		Dao<Conta> dao = new Dao<Conta>();
```

```
            try {
                  dao.adiciona(conta);
            } catch (SQLException e) {
                  return false;
            }
            return true;
      }
```

Uma conta também deve ser capaz de gerar seu próprio número, usamos um Change Method Singature;

```
public Conta criaConta(Cliente titular) throws SQLException {

            int numero = geraNumeroConta();

            if (consultaSerasa(titular.getCpf())) {

                  Calendar hoje = Calendar.getInstance();
                  Conta novaConta = new Conta(numero, hoje);
```

Podemos usar um Change Method Singature para o banco não precisar setar o titular e deixar essa responsabilidade para a Conta

```
      public Conta(int numero, Calendar dataAbertura, Cliente titular) {
            this.numero = numero;
            this.dataAbertura = dataAbertura;
            this.titular = titular;
      }
```

Com a classe Cliente também devemos separar a persistência da regra

```
      public Cliente criaCliente(String campoNome, String campoCpf) throws
SQLException {
            // recebe os dados de um formulário, cria um cliente e guarda no
banco
            // de dados
            Cliente cliente = new Cliente();
            cliente.setCpf(campoCpf);
            cliente.setNome(campoNome);

            registraCliente(cliente);

            return cliente;
      }

      public Boolean registraCliente(Cliente cliente) {
            Dao<Cliente> dao = new Dao<Cliente>();
            try {
                  dao.adiciona(cliente);
            } catch (SQLException e) {
                  e.getMessage();
                  return false;
            }

            return true;
      }
```

Pode tirar a responsabilidade do banco em setar as propriedades do cliente usando o Change Method Singature

```
      public Cliente(String nome, String cpf){
            this.nome = nome;
```

```
            this.cpf = cpf;
      }
```

E transferi os métodos da classe OperaçõesBancarias para aclasse conta, evitando violar a Lei de Demeter.

```
      public void saca(BigDecimal valor) {
            BigDecimal novoSaldo = getSaldo().subtract(valor);

            if (novoSaldo.compareTo(getLimite().negate()) >= 0) {
                  setSaldo(novoSaldo);
            } else {
                  throw new RuntimeException("Saldo insuficiente");
            }
      }

      /**
       * Deposita dinheiro em uma conta bancária
       *
       * @param valor Valor a ser sacado
       */
      public void deposita(BigDecimal valor) {
            BigDecimal novoSaldo = getSaldo().add(valor);
            setSaldo(novoSaldo);
      }
```