# Programação Orientada a Objetos
# Trabalho Prático 1

**André Lage e Augusto Mafra**

[1]Universidade Federal de Minas Gerais

## 1. Introdução

No contexto da programação, criar um sistema bancário eficiente pode se tornar uma tarefa trabalhosa: A diversidade de tipos de transações, o alto fluxo de dados e a necessidade de segurança podem ser fatores determinantes para o fracasso ou sucesso de um projeto nessa área.

Utilizando todo o conteúdo visto de orientação a objetos em sala, o objetivo do trabalho visa criar um sistema de controle bancários, no qual guaram-se informações a respeito de contas bancárias, clientes e diversos tipos de transações.

## 2. Implementação

A infraestrutura do sistema desenvolvido é modularizada em torno de duas classes principais: a classe Interface, cujos membros são todos estáticos, que lida com a interface com o usuário, e a classe Banco, que trata das operações no banco. A interface dessas classes principais é mostrada a seguir. Note que as funções do banco retornam indicadores de status (booleanos ou inteiros) para a Interface, que os avalia para emitir mensagens de erro para o usuário.

A classe banco é também a raiz da estrutura de objetos do sistema, construída por composição de objetos que representam as entidades Cliente, Conta e Movimentação.

```
1  public class Interface {
2      // Objeto banco contido na Interface
3      private static Banco banco;
4      // Membro scanner para leitura da entrada do usuario
5      private static Scanner scanner;
6      // Membro que representa o caminho do arquivo database
7      private static Path databaseFile;
8
9      public static void main(String[] args);
10
11     // Setup inicial da Interface e da linha de comando
12     private static void iniciarLinhaDeComando();
13     // Executada em loop ate o usuario executar o comando 'sair'.
           Realiza chamadas para os
14     // demais comandos.
15     private static boolean executarLinhaDeComando();
16     private static void mostrarListaDeComandos(); // implementa
           comando 'ajuda'
17     private static void cadastrarCliente(); // implementa comando
           'cadastrar'
18     private static void criarConta(); // implementa comando '
           criar_conta'
19     private static void excluirCliente(); // implementa comando '
           excluir_cliente'
```

```
20      private static void excluirConta(); // implementa comando '
            excluir_conta'
21      private static void deposito(); // implementa comando '
            deposito'
22      private static void saque(); // implementa comando 'saque'
23      private static void cobrarTarifa(); // implementa comando '
            cobrar_tarifa'
24      private static void cobrarCPMF(); // implementa comando '
            cobrar_cpmf'
25      private static void saldo(); // implementa comando 'saldo'
26      private static void extrato(); // implementa comando 'extrato
            '
27      private static void listarClientes(); // implementa comando '
            listar_clientes'
28      private static void listarContas(); // implementa comando '
            listar_contas'
29
30      // salva dados do banco no arquivo databaseFile.
31      // Executado quando o usuario executa 'sair'.
32      private static void salvar();
33      // constroi objeto banco a partir do arquivo databaseFile.
34      // Executado sempre que a linha de comando e iniciada.
35      private static void restaurar();
36
37      // Funcoes auxiliares para requisitar inputs do usuario
38      // Mostram mensagem no terminal e recebem input no formato
            correspondente
39      private static String promptString(String mensagem);
40      private static int promptInt(String mensagem);
41      private static double promptDouble(String mensagem);
42      private static GregorianCalendar promptCalendar(String
            mensagem);
43  }
```

A classe banco inclui as funcoes de incialização, salvar e restaurar a árvore de objetos composta. Ela oferece ainda os métodos de cadastro e exclusão de itens do banco de dados e para realização de operações bancárias, tais como saques, depósitos e transferências. O seu acesso é inteiramente controlado pela classe interface, e nenhuma interação com o console é realizada diretamente na classe Banco.

Destaca-se, ainda, a necessidade de se garantir o encapsulamento nas funções de acesso ao banco. Essa propriedade é assegurada pelo retorno de cópias dos objetos nas funções getCliente, getClientes e getContas. A função getConta requer a possibilidade de se manipular o objeto retornado, e mantém por isso uma referência direta para o item obtido. Ela é mantida como acesso privado por causa disso.

```
1   public class Banco {
2       private String nomeBanco;
3       private ArrayList<Cliente> clientes; // Clientes contidos no
            banco
4       private ArrayList<Conta> contas;    // Contas contidas no
            banco
5
6       // Construtor por nome
7       public Banco(String nome);
```

```
8       // Construtor pelo arquivo no filePath
9       public Banco(Path filePath);
10      public boolean salvar(Path filePath);
11
12      private void restauraBanco(List<String> data, ListIterator<
            String> i);
13      private void restauraClientes(List<String> data, ListIterator
            <String> i);
14      private void restauraContas(List<String> data, ListIterator<
            String> i);
15      private ArrayList<Movimentacao> restauraMovimentacoes(List<
            String> data, ListIterator<String> i);
16      private void restauraConta(int numConta, double saldo,
            ArrayList<Movimentacao> movimentacoes);
17      private void salvaClientes(List<String> data);
18      private void salvaContas(List<String> data);
19
20      public boolean addCliente(Cliente cliente);
21      public Cliente getCliente(String cpf_cnpj); // Retorna copia
            do cliente com o CPF/CNPJ
22      public ArrayList<Cliente> getClientes(); // Retorna copia de
            clientes
23      public ArrayList<Conta> getContas(); // Retorna copia de
            contas
24      // Retorna referencia para conta com o numConta.
25      // Mantida privada para impedir acesso a referencias para
            Contas.
26      private Conta getConta(int numConta);
27
28      public Conta criaConta(Cliente cliente);
29
30      public int removeCliente(String cpf);
31      public boolean removeConta(int numConta);
32
33      public boolean deposito(int numConta, double valor);
34      public boolean saque(int numConta, double valor);
35      public ArrayList<Movimentacao> extrato(int numConta);
36      public ArrayList<Movimentacao> extrato(int numConta,
            GregorianCalendar dataInicial);
37      public ArrayList<Movimentacao> extrato(int numConta,
            GregorianCalendar dataInicial, GregorianCalendar
            dataFinal);
38      public void tarifa();
39      public void cpmf();
40      public double saldo(int numConta);
41      public int transferencia(int numContaOrigem, int
            numContaDestino);
42  }
```