Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Encapsulamento das operações de persistência em banco de dados

NOTAS DE AULA - Teoria 08

Linguagem de Programação II
2º semestre de 2015
Versão 2.0

Objetivos

- Definir, modelar e implementar classes que encapsulam as operações de persistência em banco de dados.
- Implementar as operações CRUD (*Create*, *Read*, *Update*, *Delete*).

Referência

A referência para esta aula é o capítulo 8 de:

> Alur, D.; Crupi J.; Malks, D. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall, 2003. (versão on-line disponível em http://www.oracle.com/technetwork/java/dataaccessobject-138824.html)

Observações:

- As notas de aula são material de apoio para estudo e não têm o objetivo de apresentar o assunto de maneira exaustiva. Não deixe de ler o material de referência da disciplina.
- Para reduzir o número de linhas de código, os exemplos apresentados omitem propositalmente a importação das classes. Para esta aula, a maior parte das classes pertencem ao pacote **java.sql**. As principais IDEs Java possuem recursos para auxiliar a inclusão das importações das classes.
- Os trechos de código fornecidos são simplistas e curtos para podermos focar no assunto que está sendo estudado, por isso não devem ser utilizados diretamente em produção.

Introdução

Os dados das corporações podem ser armazenados em **diferentes tipos de repositórios**.

- Sistema de arquivos
- Base de dados relacional
- Base de dados orientada a objetos
- Mainframes

Dificuldades trazidas pelo uso de diferentes tipos de repositórios

- Cada tipo de repositório fornece funcionalidades e mecanismos de acesso (APIs) distintos.
- Se a lógica da aplicação tiver uma dependência direta dos mecanismos de acesso, migrar uma aplicação para utilizar outro tipo de repositório pode ser difícil.

O que precisamos?

- Desacoplar a implementação da persistência do resto da aplicação.
- Oferecer uma API única de acesso aos dados, independente do tipo de repositório de dados.
- Organizar a lógica de acesso e encapsular as funcionalidades proprietárias.

Aula passada

tabela **contas**

nro_conta	saldo
123	500,00
124	1000,00
125	2500,00

tabela **titulares**

nro_titular	nome	rg	cpf
2	Marcos Antônio	22333444	09988877765
3	Rosimeire Aparecida	11222333	08866655543
4	Roberto Carlos	33444555	07755544432
5	André Barros	44555666	06655533321

tabela **contas_titulares**

nro_conta	nro_titular
123	2
124	3
124	5
125	4

Exemplo de aplicação que faz a consulta de todas as contas

```
public class AppSelectContas {

    public static void main(String[] args)
            throws ClassNotFoundException, SQLException {

        Class.forName("org.apache.derby.jdbc.ClientDriver");
        Connection conexao;
        String url = "jdbc:derby://127.0.0.1:1527/banco";
        String usuario = "app";
        String senha = "app";
        conexao = DriverManager.getConnection(url, usuario, senha);

        Statement st;
        st = conexao.createStatement();

        String sql = "SELECT nro_conta, saldo FROM contas";
        ResultSet resultados = st.executeQuery(sql);

        System.out.println("Dados das contas:");
        while (resultados.next()) {
            System.out.print("Número: " + resultados.getLong("nro_conta") + " - ");
            System.out.println("Saldo: R$ " + resultados.getBigDecimal("saldo"));
        }

        conexao.close();
    }

}
```

Solução

Utilizar o padrão DAO (*Data Access Object*).

Classe **Conta**

```
public class Conta {
    private long numero;
    private BigDecimal saldo;

    // construtores, getters e setters
}
```

Classe Titular

```
public class Titular {
    private long numero;
    private String nome;
    private String rg;
    private String cpf;

    // construtores, getters e setters
}
```

Operações de acesso aos dados das contas

```
public interface ContaDaoInterface {
    List<Conta> listarTudo();

    // outras operações
}
```

Operações de acesso aos dados dos titulares

```
public interface TitularDaoInterface {
    List<Titular> listarTudo();

    // outras operações
}
```

Implementação das operações

```
public class ContaDaoJavaDb implements ContaDaoInterface {
    @Override
    public List<Conta> listarTudo() {
        List<Conta> contas = new ArrayList<>();
        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            String url = "jdbc:derby://127.0.0.1:1527/banco";
            String usuario = "app";
            String senha = "app";
            Connection conexao = DriverManager.getConnection(url, usuario, senha);
            Statement st = conexao.createStatement();
            String sql = "SELECT nro_conta, saldo FROM contas";
            ResultSet resultados = st.executeQuery(sql);
            while (resultados.next()) {
                Conta c = new Conta(resultados.getLong("nro_conta"),
                    resultados.getBigDecimal("saldo"));
                contas.add(c);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException();
        }
        return contas;
    }
}
```

Exemplo de Utilização

Aplicação que faz acesso aos dados de todas as contas (com DAO)

```
public class AppSelectContasComDao {
      public static void main(String[] args) {
        ContaDaoInterface dao = new ContaDaoJavaDb();
        List<Conta> todasContas = dao.listarTudo();
        for (Conta c: todasContas) {
            System.out.print("Número: " + c.getNumero() + " - ");
            System.out.println("Saldo: R$ " + c.getSaldo());
        }
    }
}
```

Aplicação que faz acesso aos dados de todas as contas (sem DAO)

```
public class AppSelectContas {
    public static void main(String[] args)
            throws ClassNotFoundException, SQLException {
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        Connection conexao;
        String url = "jdbc:derby://127.0.0.1:1527/banco";
        String usuario = "app";
        String senha = "app";
        conexao = DriverManager.getConnection(url, usuario, senha);
        Statement st;
        st = conexao.createStatement();
        String sql = "SELECT nro_conta, saldo FROM contas";
        ResultSet resultados = st.executeQuery(sql);
        System.out.println("Dados das contas:");
        while (resultados.next()) {
            System.out.print("Número: " + resultados.getLong("nro_conta") + " - ");
            System.out.println("Saldo: R$ " + resultados.getBigDecimal("saldo"));
        }
        conexao.close();
    }
}
```

Adição de outras operações

Outras operações

```
public interface ContaDaoInterface {
    List<Conta> listarTudo();

    // obtém uma lista das contas de um titular
    List<Conta> listarContasDe(Titular titular);

    // gravar uma nova conta
    void salvarNova(Conta conta);

    // obtém a instância de Conta correspondente ao número informado
    Conta buscar(long nroConta);

    // mais operações
}
```

```
public interface TitularDaoInterface {
    List<Titular> listarTudo();

    // obtém uma lista dos titulares de uma conta
    List<Titular> listarTitularesDe(Conta conta);

    // obtém a instância de Titular correspondente ao número informado
    Titular buscar(long nroTitular);

    // mais operações
}
```

Outras melhorias

Separação do gerenciamento de conexões

Os comandos SQL se aplicam às bases de dados relacionais, mas a nossa classe **ContaDaoJavaDb** só cria conexões com o Java DB.

Para separar o gerenciamento de conexões, podemos declarar uma nova interface

```
interface ConexaoInterface {
    Connection getConnection();
    void close();
}
```

Separação do gerenciamento de conexões (cont.)

A implementação para a conexão com o Java DB ficaria

```
public class ConexaoJavaDb implements ConexaoInterface {
    private String usuario;
    private String senha;
    private String hostname;
    private int porta;
    private String nomeBancoDados;
    private Connection conexao;
    public ConexaoJavaDb(String usuario, String senha, String hostname,
                         int porta, String nomeBancoDados) {
        this.usuario = usuario;
        this.senha = senha;
        this.hostname = hostname;
        this.porta = porta;
        this.nomeBancoDados = nomeBancoDados;
    }
```

(continua...)

(continuação)

```
    @Override
    public Connection getConnection() {
        if (conexao == null) {
            try {
                Class.forName("org.apache.derby.jdbc.ClientDriver");
                String url = "jdbc:derby://" + hostname + ":" +
                      porta + "/" + nomeBancoDados;
                conexao = DriverManager.getConnection(url, usuario, senha);
            } catch (Exception ex) {}
        }
        return conexao;
    }
    @Override
    public void close() {
        try {
            conexao.close();
        } catch (SQLException ex) {}
        conexao = null;
    }
}
```

Separação do gerenciamento de conexões (cont.)

A classe **ContaDaoJavaDb** seria substituída pela classe **ContaDaoRelacional**, que não possui nenhuma referência direta às particularidades do Java DB:

```
public class ContaDaoRelacional implements ContaDaoInterface {
    private ConexaoInterface conexao;
    public ContaDaoRelacional(ConexaoInterface conexao) {
        this.conexao = conexao;
    }
```

(continua...)

(continuação)

```
    @Override
    public List<Conta> listarTudo() {
        List<Conta> contas;
        contas = new ArrayList<>();
        try {
            Statement st;
            st = conexao.getConnection().createStatement();
            String sql = "SELECT nro_conta, saldo FROM contas";
            ResultSet resultados = st.executeQuery(sql);
            while (resultados.next()) {
                long n = resultados.getLong("nro_conta");
                BigDecimal b = resultados.getBigDecimal("saldo");
                Conta c = new Conta(n, b);
                contas.add(c);
            }
            conexao.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        throw new RunTimeException();
        }
        return contas;
    }
}
```

Separação do gerenciamento de conexões (cont.)

Utilizando esta nova interface **ConexaoInterface** e as classes **ConexaoJavaDb** e **ContaDaoRelacional**, o programa que lista todas as contas ficaria

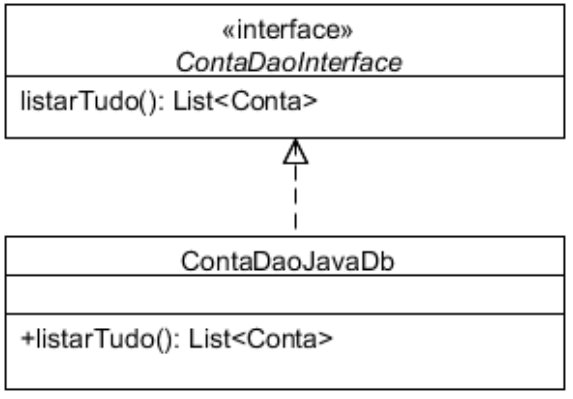
```
public class AppSelectContasComDao {
    public static void main(String[] args) {
        ConexaoInterface conexao = new ConexaoJavaDb("app", "app",
                                                     "127.0.0.1", 1527, "banco");
        ContaDaoInterface dao;
        dao = new ContaDaoRelacional(conexao);
        List<Conta> todasContas;
        todasContas = dao.listarTudo();
        for (Conta c: todasContas) {
            System.out.print("Nro: " + c.getNumero());
            System.out.print(" - ");
            System.out.println("Saldo: R$ " + c.getSaldo());
        }
    }
}
```

Para aplicações que utilizam várias conexões simultâneas (por exemplo, aplicações web) é necessário algo que gerencie um *pool* de conexões, mas isto não faz parte do escopo desta aula.

Outras implementações da interface **ContaDaoInterface**

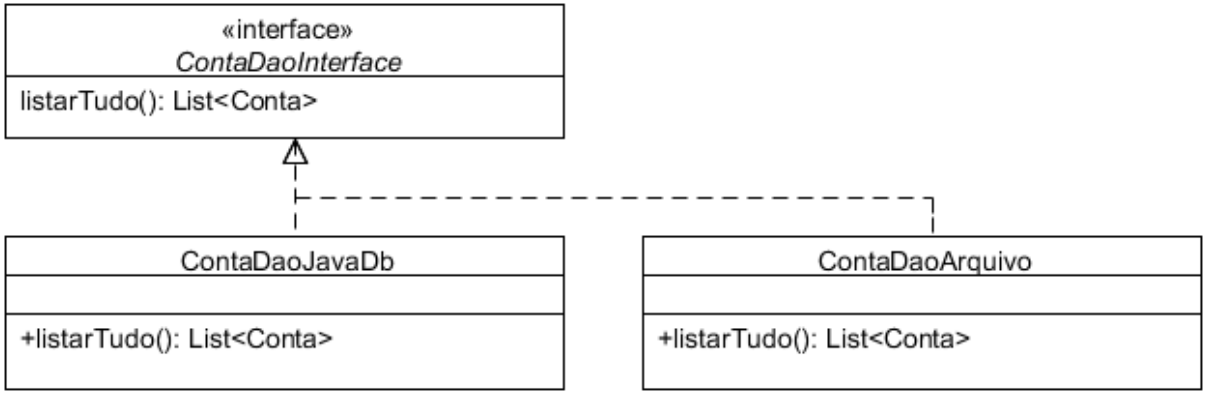
ContaDaoJavaDb implementa a interface **ContaDaoInterface**

Nos nossos exemplos, a interface **ContaDaoInterface** foi implementada pela classe **ContaDaoJavaDb**.



ContaDaoArquivo também pode implementar **ContaDaoInterface**

Podemos desenvolver **ContaDaoArquivo**, uma classe que também implementa **ContaDaoInterface** mas acessa as informações que estão armazenadas em um arquivo.



Código de **ContaDaoArquivo**

```java
public class ContaDaoArquivo implements ContaDaoInterface {
  @Override
  public List<Conta> listarTudo() {
    List<Conta> contas = new ArrayList<>();
    try (BufferedReader br = new BufferedReader(new FileReader("contas.txt"))) {
      String linha;
      while ( (linha = br.readLine()) != null ) {
        String campos[] = linha.split(";");
        int nroConta = Integer.parseInt(campos[0]);
        BigDecimal saldo = new BigDecimal(campos[1]);
        Conta conta = new Conta(nroConta, saldo);
        contas.add(conta);
      }
    } catch (FileNotFoundException ex) {
      System.out.println("Arquivo contas.txt não encontrado!");
    } catch (IOException ex) {
      System.out.println("Erro na leitura do arquivo!");
    }
    return contas;
  }
}
```

Troca de implementações de **ContaDaoInterface**

Vimos anteriormente o código de um programa que listava as contas armazenadas na base de dados Java DB.

```
public class AppSelectContasComDao {
      public static void main(String[] args) {
        ContaDaoInterface dao = new ContaDaoJavaDb();
        List<Conta> todasContas = dao.listarTudo();
        for (Conta c: todasContas) {
            System.out.print("Número: " + c.getNumero() + " - ");
            System.out.println("Saldo: R$ " + c.getSaldo());
        }
    }
}
```

Para listar as contas armazenadas no arquivo **contas.txt**, basta trocar a instanciação de **ContaDaoJavaDb** pela de **ContaDaoArquivo**.

```
public class AppSelectContasComDao {
      public static void main(String[] args) {
        ContaDaoInterface dao = new ContaDaoArquivo();
        List<Conta> todasContas = dao.listarTudo();
        for (Conta c: todasContas) {
            System.out.print("Número: " + c.getNumero() + " - ");
            System.out.println("Saldo: R$ " + c.getSaldo());
        }
    }
}
```

Listagem das contas utilizando qualquer implementação de **ContaDaoInterface**

No código abaixo, note que o método **apresentarContas** é capaz de apresentar as informações das contas utilizando qualquer implementação de **ContaDaoInterface**.

```
public class AppSelectContasComDao {
    public static void apresentarContas(ContaDaoInterface dao) {
        List<Conta> todasContas;
        todasContas = dao.listarTudo();
        for (Conta c: todasContas) {
            System.out.print("Nro: " + c.getNumero());
            System.out.print(" - ");
            System.out.println("Saldo: R$ " + c.getSaldo());
        }
    }
    public static void main(String[] args) {
        apresentarContas(new ContaDaoArquivo());
        apresentarContas(new ContaDaoJavaDb());
    }
}
```

Obrigado!