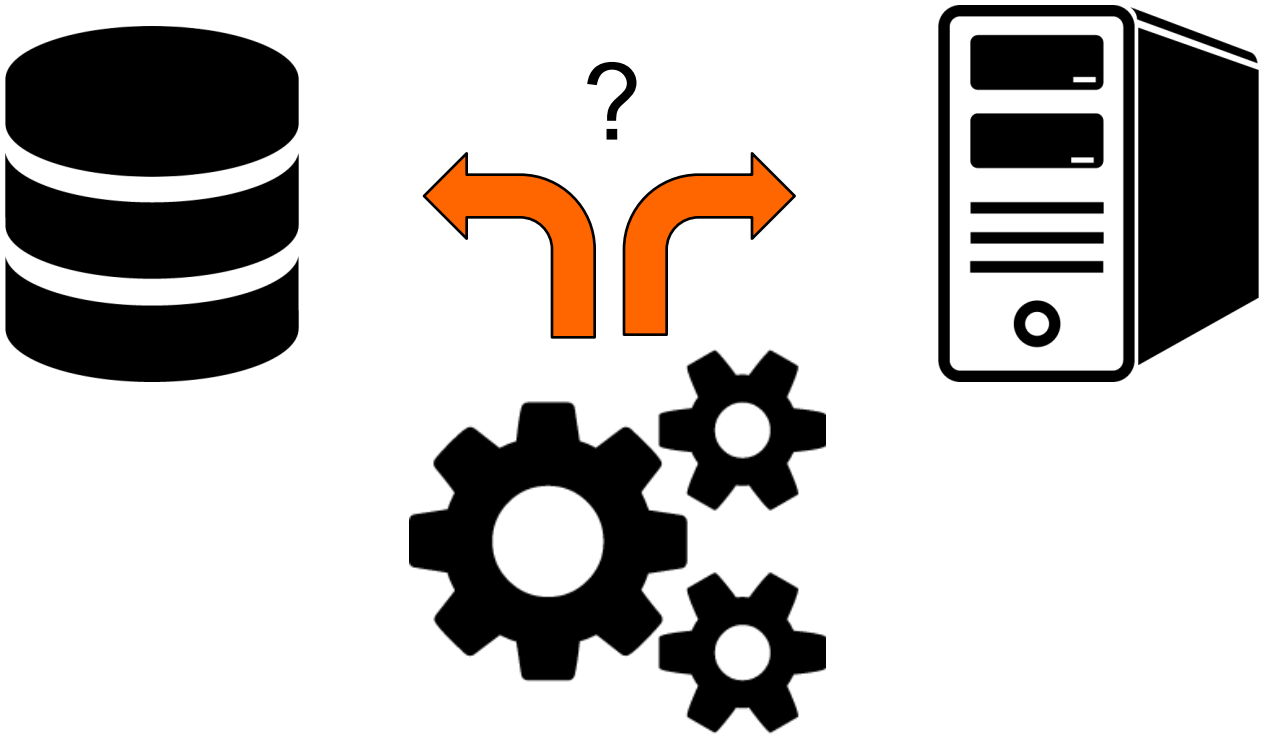
Desenvolvimento de Sistemas Software

Aula Teórica 26: Comunicação com bases de dados 2 - DAO

Desenvolvimento de Sistemas Software
José Creissac Campos / António Nestor Ribeiro / Rui Couto

Contexto

- Existem duas visões distintas acerca da persistência em aplicações
- lógica na aplicação vs lógica na base de dados



ORM

- Torna a forma de persistência transparente para o utilizador.
- Fornece API consistente para acesso a objetos.
- Fortalece a independência entre camada de dados e camada de negócio.

ORM – DAO vs ResultSet

- DAO (Data Access Object) simplifica o acesso aos dados.
- Permite nos carregar para memória, de forma transparente, instâncias persistidas na base de dados.
- Usamos as entidades de forma previsível.

ORM – DAO vs ResultSet

- Código para obter nome do utilizador.
  - Com ResultSet

```
//ResultSet
String nome;
ResultSet rs = getResultSet(id);
if(rs.next()) {
    nome = rs.get("nome");
}
```

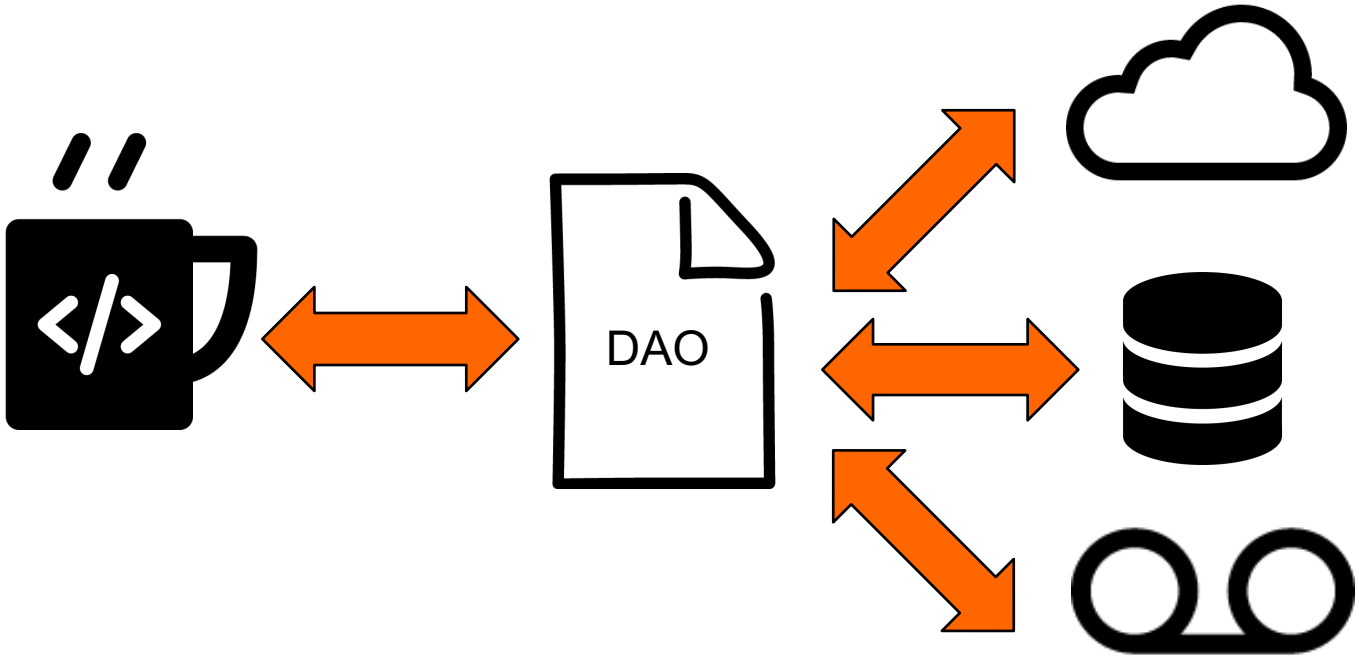
  - Com DAO

```
//DAO
User u = UserDAO.get(id);
String nome = u.getNome();
```

- Na prática (para este exemplo), DAO necessita de menos 50% de linhas de código

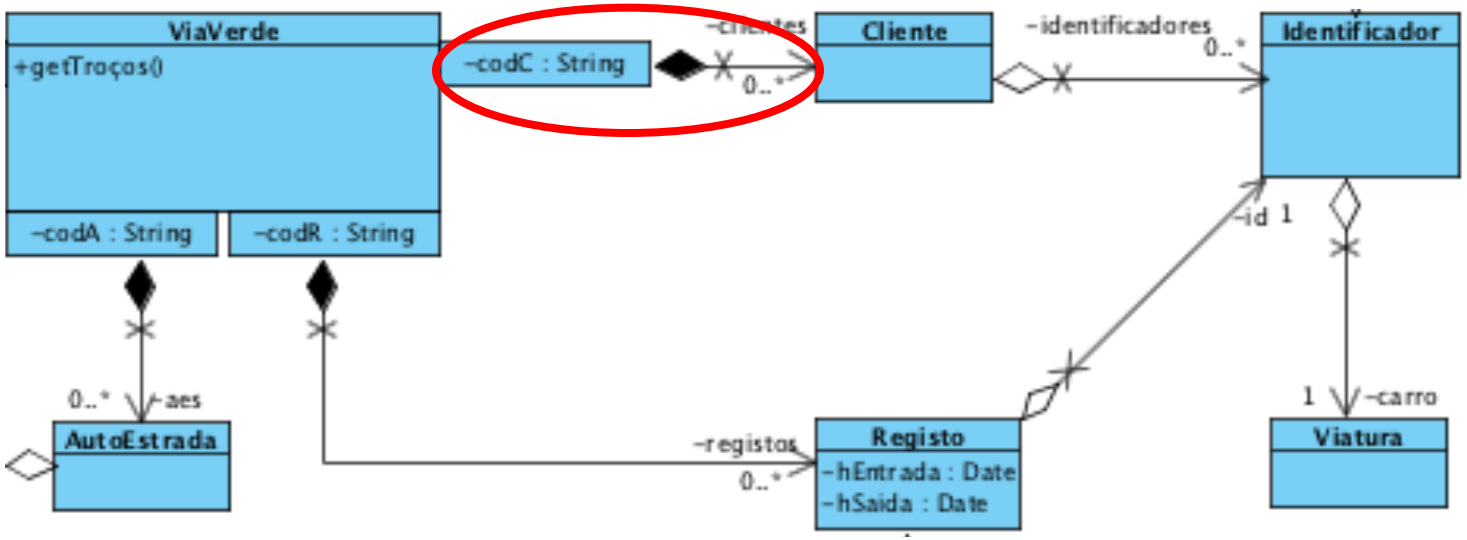
ORM

- Permite reduzir o código necessário para fazer leituras.
- Oculta detalhes de implementação (*queries*, ligações, etc.).
- Interface comum permite persistência evoluir de forma independente.



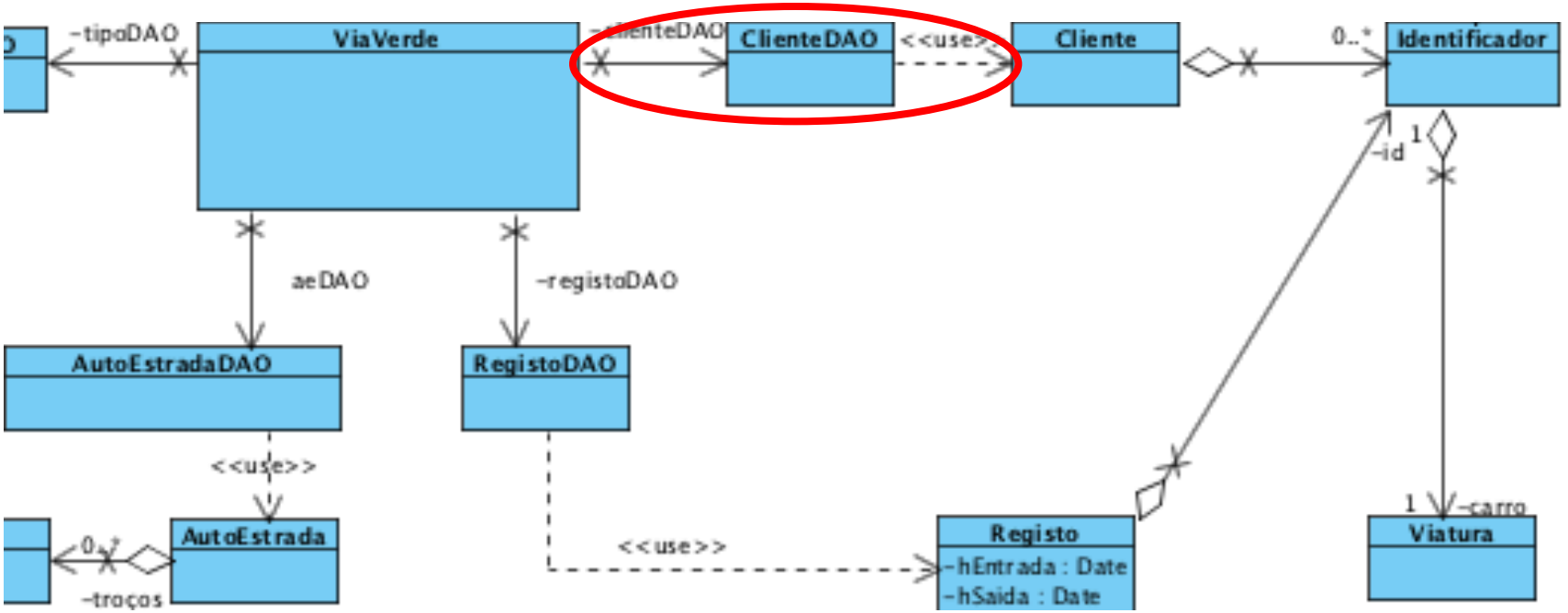
DAO – Abordagem

- Composição denota um relacionamento "forte" entre classes.
- I.e., `ViaVerde` é **composto** por (entre outros) **Cliente**.
- Assim, `ViaVerde` tem necessidade de gerir os clientes.



DAO – Abordagem (2)

- Propõe-se que essas associções sejam substituídas por DAOs.
- DAOs vão fazer a gestão das instâncias do tipo respectivo.
- Vão também substituir os mapeamentos.



DAO – Abordagem (3)

- DAOs são quem faz a comunicação com a base de dados.
- São o ponto de comunicação entre a solução de persistência, e as entidades no código.
- São necessários (no mínimo) DAOs para as classes que compõem a solução:
  - `Cliente` → `ClienteDAO`
  - `Registo` → `RegistoDAO`
  - `AutoEstrada` → `AutoEstradaDAO`

DAO – Implementação

- Devem ser consistentes entre si.
  - Manter uma interface “previsível” e comum.
  - Forcener métodos realmente úteis.
  - Desenhar solução de forma modular, precavendo crescimento da mesma.
  - Manter coerência com código existente (?).
- Existem diversas opções para garantir essas propriedades.

DAO – Implementação

- Ter métodos adequados a cada classe:

```
getUtilizador(String id)

getUtilizadorPorNome(String nome)
```

  - É solução feita por medida.
  - Não garante uniformidade no processo.
  - Não garante compatibilidade com código prévio.

DAO – Implementação

- Definir uma interface de acesso comum:

```
getById(String id)

getByName(String name)

listAll()
```

.Fornece interface feito por medida, mas comum.

.Pode ser difícil garantir uma interface comum.

.Não garante compatibilidade com código prévio.

DAO – Implementação

- Implementar uma interface já bem conhecida: `Map`

```
get(Object key)

put(Object key, Object value)

...
```

.Garante uniformidade, a interface é bem conhecida

.Garante compatibilidade com o que tínhamos antes (mapeamentos).

- Map define os métodos que necessitamos de implementar.

DAO – Implementação

- Solução passa for fazer os nossos DAOs implementar a interface

```
Map<TipoChave, TipoObjecto>
```

- Por exemplo:

```
Public class ClienteDAO implements Map<String, Cliente>
```

DAO – Implementação

- Passamos a ter de implementar em **UtilizadorDAO**:

```
public int size() {
public boolean isEmpty();
public boolean containsKey(Object key);
public boolean containsValue(Object value);
public Cliente get(Object key);
public Cliente put(String key, Cliente value);
public Cliente remove(Object key);
public void putAll(Map<? extends String, ? extends Cliente> m);
public void clear();
public Set<String> keySet();
public Collection<Cliente> values();
public Set<Entry<String, Cliente>> entrySet();
```

DAO – Exemplo

- Classe ViaVerde com Map:

```
public class ViaVerde {
    private Map<String,Cliente> clientes;
    //...

    public List<String> getAllNames() {
        List<String> list = new ArrayList<String>();
        Collection<Cliente> ccl = clientes.values();
        for(Cliente c : ccl) {
            list.add(c.getName());
        }
        return list;
    }

    //...

}
```

DAO – Exemplo

- Base de dados:

```
CREATE TABLE `cliente` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(45) DEFAULT NULL,
  `nif` int(11) DEFAULT NULL,
  `datanascimento` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
```

- Vamos precisar de uma tabela para cada classe a persistir
  - Independentemente da existência ou não de um DAO para essa classe.

DAO – Exemplo

- Classe Connect:

```
public class Connect {

    //Responsabilidade de fechar a ligação fica do lado de quem invoca
    public static Connection connect()
            throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.jdbc.Driver");
        Connection connect =
DriverManager.getConnection("jdbc:mysql://host/viaverde?user=username&password=pas
sword");
        return connect;
    }
}
```

DAO – Exemplo

- Classe ClienteDAO:

```
public class ClienteDAO implements Map<String, Cliente> {

    @Override
    public int size() {
        int size = -1;
        Connection con = null;
        try {
            con = Connect.connect();
            PreparedStatement ps = con.prepareStatement("select count(id) from
cliente");
            ResultSet rs = ps.executeQuery();
            if(rs.next()) {
                size = rs.getInt(1);
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            try {
                con.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return size;
    }
```

DAO – Exemplo

```
@Override
public Cliente get(Object key) {
    Cliente c = null;
    Connection con = null;
    try {
        con = Connect.connect();
        PreparedStatement ps = con.prepareStatement("select * from cliente where id = ?");
        ps.setInt(1, Integer.parseInt(key.toString()));
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            c = new Cliente(rs.getInt("id"),rs.getString("nome"),
rs.getDate("datanascimento"), rs.getInt("nif"));
        }
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        try {
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return c;
}
```

DAO – Exemplo

```
@Override
public Collection<Cliente> values() {
    Collection<Cliente> res = new ArrayList<>();
    Cliente c = null;
    Connection con = null;
    try {
        con = Connect.connect();
        PreparedStatement ps = con.prepareStatement("select * from cliente");
        ResultSet rs = ps.executeQuery();
        while(rs.next()) {
            c = new Cliente(rs.getInt("id"),rs.getString("nome"),
rs.getDate("datanascimento"), rs.getInt("nif"));
            res.add(c);
        }
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    } finally {
        try {
            con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return res;
}
```

DAO – Exemplo

- Classe ViaVerde com DAO:

```
public class ViaVerde {
    private ClienteDAO clientes;
    //...

    public List<String> getAllNames() {
        List<String> list = new ArrayList<String>();
        Collection<Cliente> ccl = clientes.values();
        for(Cliente c : ccl) {
            list.add(c.getName());
        }
        return list;
    }

    //...

}
```

DAO – Exemplo

- O que fazer com restantes classes?

- (Pelo menos) duas abordagens possíveis.
- Carregar dados por cascada:
  - Ao carregar uma instância (e.g. Cliente), carregar todos os restantes dados que ele agrega (e.g. Identificadores).
- Um DAO por classe:
  - Replicar na camada de dados todas as classes que necessitem de persistência.

DAO – Discussão

- Substituir Map por DAO é processo praticamente transparente.
- Map continua a funcionar da mesma forma, implementação é abstraída.
- Código dos DAO tenderá a ser semelhante entre eles.
- Transacções podem existir dentro dos DAO.
  - DAOs podem e devem conter alguma lógica de negócio, sempre que justificável.

DAO – Discussão

- Lógica de negócio *deve* ser puxada para camada de negócio.
- Seguindo a ideologia ORM, base de dados serve apenas para depositar a buscar dados.
- Não é no entanto proibida a existência de alguma lógica de dados, ao nível das queries.
- Essa lógica deve ser transparente ao utilizador, e não causar comportamentos inesperados.
  - E.g.: validação do NIF na base de dados

DAO – Feedback

- Métodos de `Map` não possuem excepções.
- (Pelo menos) duas opções para obter feedback.
- Verificação normal de dados:
  - Map "normal" também não reporta excepções – necessidade de lidar com **null**.
  - Não dá informação sobre o que aconteceu na base de dados.

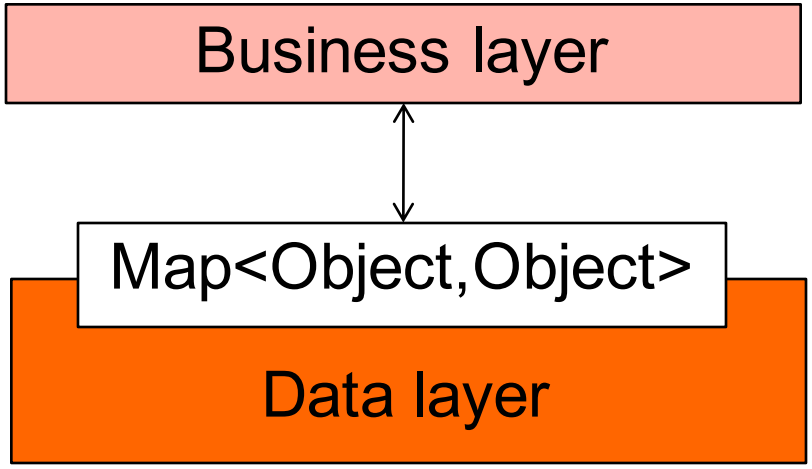
DAO – Feedback

- Usar excepções de *runtime*
  - São excepções que não têm de ser declaradas na assinatura do método:

    ```
    throw new RuntimeException(mensagem)
    ```
  - Permitem fornecer mais informação sobre o erro.
  - Necessitam de cuidado, porque compilador não obriga a apanhar essas excepções.

DAO – Facade

- A interface Map define a façade para a camada de dados.
- No entanto existem diversas implementações (de acordo com o objecto em questão).
- Objectivo do façade é mantido: uma interface que abstrai uma camada, e ponto único de comunicação para a camada superior.



DAO – Casos excepcionais

- Pode fazer sentido não implementar alguns métodos.
  - Soluções:
    - Lançar excepção: há que ter atenção ao usar.
    - Retornar nulls: há que perceber o que está a fazer.
    - Implementar: garante consistencia, implica mais trabalho.

DAO – Casos excepcionais

- Pode ser necessário, ou extremamente vantajoso ter algum tipo de lógica na base de dados
  - Deve ser usado excepcionalmente.
  - Nunca deve ser usado quando a solução na camada de negócios seja tão ou mais simples que a solução na base de dados.
  - Chaves únicas, chaves estrangeiras, etc. não é considerado lógica de negócio.

Sumário

- Persistência deve passar a ser feita por DAOs.
- DAOs substituem mapeamentos, mantendo o mesmo interface.
- Interface a implementar é Map.
- Lógica de negócio deve existir na camada de negócio, e eventualmente alguma lógica de dados nos DAOs
- Preocupações com a base de dados (prepared statements, transactions, etc.) existem também nos DAOs.