Curso: Programação Orientada a Objetos com Java

http://educandoweb.com.br

Prof. Dr. Nelio Alves

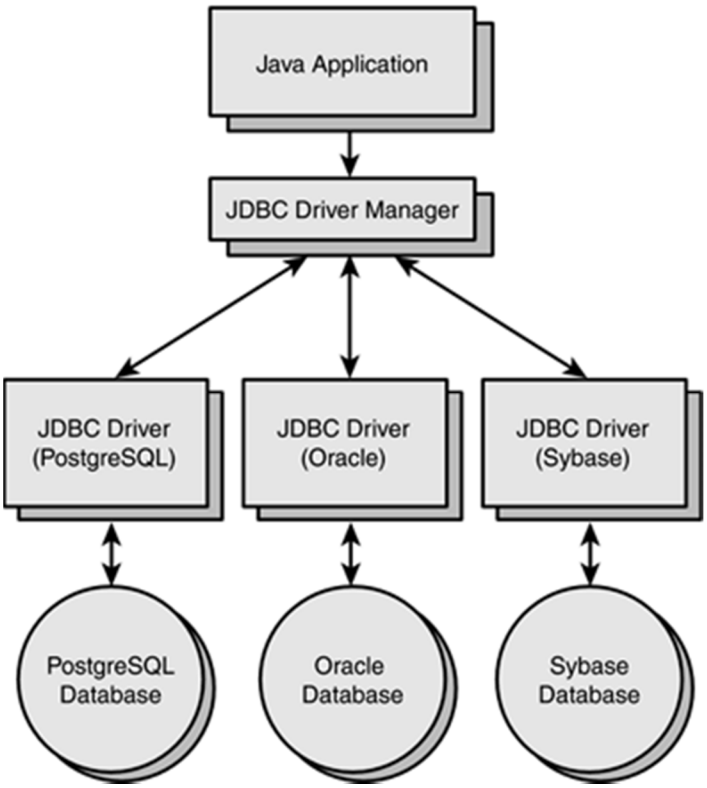
Capítulo: Acesso a banco de dados com JDBC

Objetivo geral:

- Conhecer os principais recursos do JDBC na teoria e prática
- Elaborar a estrutura básica de um projeto com JDBC
- Implementar o padrão DAO manualmente com JDBC

Visão geral do JDBC

- JDBC (Java Database Connectivity): API padrão do Java para acesso a dados
- Páginas oficiais:
  - https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
  - https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html
- Pacotes: **java.sql** e **javax.sql** (API suplementar para servidores)



Instalação das ferramentas:

- Instalar o MySQL Server e o MySQL Workbench

Preparação do primeiro projeto no Eclipse

Código fonte: https://github.com/acenelio/jdbc1

Checklist:

- Usando o MySQL Workbench, crie uma base de dados chamada "coursejdbc"
- Baixar o MySQL Java Connector
- Caso ainda não exista, criar uma User Library contendo o arquivo .jar do driver do MySQL
  - Window -> Preferences -> Java -> Build path -> User Libraries
  - Dê o nome da User Library de MySQLConnector
  - Add external JARs -> (localize o arquivo jar)
- Criar um novo Java Project
  - Acrescentar a User Library MySQLConnector ao projeto
- Na pasta raiz do projeto, criar um arquivo "db.properties" contendo os dados de conexão:

```
user=developer
password=1234567
dburl=jdbc:mysql://localhost:3306/coursejdbc
useSSL=false
```

- No pacote "db", criar uma exceção personalizada DbException
- No pacote "db", criar uma classe DB com métodos estáticos auxiliares
  - Obter e fechar uma conexão com o banco

Demo: recuperar dados

Script SQL: material de apoio ou https://github.com/acenelio/demo-dao-jdbc/blob/master/database.sql

Código fonte: https://github.com/acenelio/jdbc2

API:

- Statement
- ResultSet
  - first() *[move para posição 1, se houver]*
  - beforeFirst() *[move para posição 0]*
  - next() *[move para o próximo, retorna false se já estiver no último]*
  - absolute(int) *[move para a posição dada, lembrando que dados reais começam em 1]*

Checklist:

- Usar o script SQL para criar a base de dados "coursejdbc"
- Fazer um pequeno programa para recuperar os departamentos
- Na classe DB, criar métodos auxiliares estáticos para fechar ResultSet e Statement

Atenção: o objeto ResultSet contém os dados armazenados na forma de tabela:

(posição)	Id	Name
1	1	Computers
2	2	Electronics
3	3	Fashion
4	4	Books

Demo: inserir dados

Código fonte: https://github.com/acenelio/jdbc3

API:

- PreparedStatement objeto que permite montar a consulta sql deixando os parametros para por depois
- executeUpdate
- Statement.RETURN_GENERATED_KEYS
- getGeneratedKeys

Checklist:

- Inserção simples com preparedStatement
- Inserção com recuperação de Id

Demo: atualizar dados

Código fonte: https://github.com/acenelio/jdbc4

Demo: deletar dados

Código fonte: https://github.com/acenelio/jdbc5

Checklist: Criar uma exceção personalizada, pq é muito comum ao deletar algo do bando aparecer um problema de integridade referencial

- Criar DbIntegrityException
- Tratar a exceção de integridade referencial

Demo: transações Operação que tem que manter a consistência do banco de dados

Referências: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/product-overview/acid.html

Código fonte: https://github.com/acenelio/jdbc6

API:

- setAutoCommit(false) Cada operação que fizer será confirmada automaticamente se verdadeiro
- commit() Confirmar a operação
- rollback() Desfazer a operação

4 propriedades que deve ter
- Atômica - Ou acontece tudo ou não acontece nada
- Consistente
- Isolada
- Durável

Exemplo clássico - transferência bancária

Padrão de projeto DAO (Data Access Object)

Objeto de acesso a dados

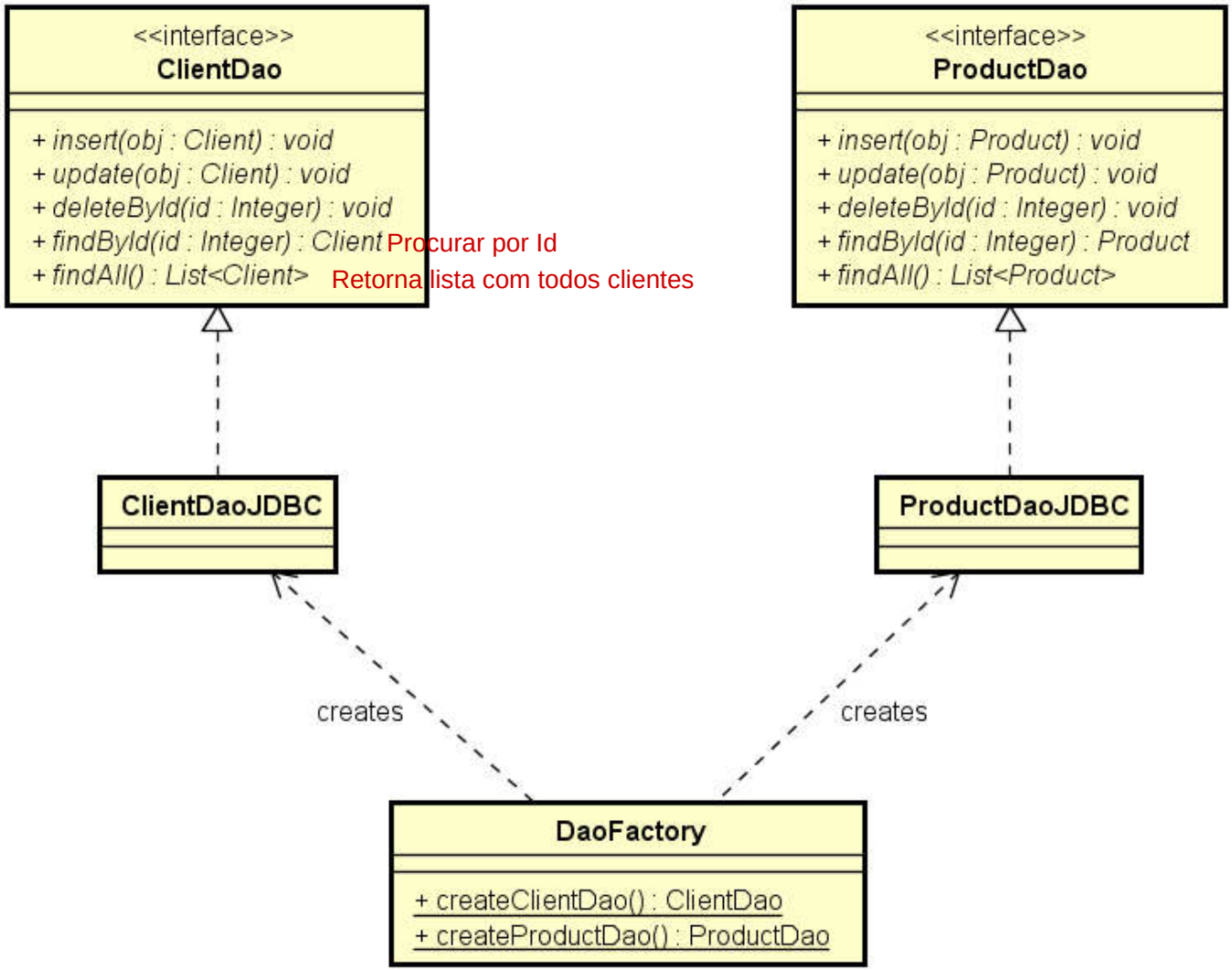
Referências:

https://www.devmedia.com.br/dao-pattern-persistencia-de-dados-utilizando-o-padrao-dao/30999

https://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Ideia geral do padrão DAO:

- Para cada entidade, haverá um objeto **responsável por fazer acesso a dados relacionado a esta entidade**. Por exemplo:
  - Cliente: ClienteDao
  - Produto: ProdutoDao
  - Pedido: PedidoDao
- Cada DAO será definido por uma interface.
- A injeção de dependência pode ser feita por meio do padrão de projeto Factory



Creating project and git repository

Project: http://github.com/acenelio/demo-dao-jdbc

Checklist:

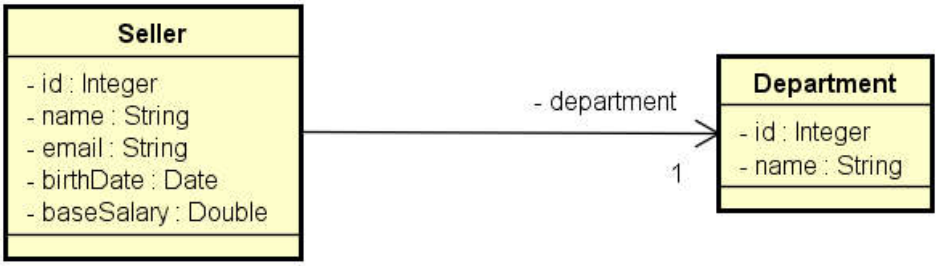
- Github: create a new project
  - **NOTE:** choose **`.gitignore`** type as Java
- Eclipse: create new java project with MySQLConnector library
  - Copy db package and db.properties from: https://github.com/acenelio/jdbc5
- Create local repository and push to Github:

```
git init
git remote add origin https://github.com/acenelio/jdbc-dao-demo.git
git pull origin master
git add .
git commit -m "Project created"
git push -u origin master
```

Department entity class

Entity class checklist:

- Attributes
- Constructors
- Getters/Setters
- hashCode and equals
- toString
- implements Serializable



Seller entity class

(video)

DepartmentDao and SellerDao interfaces

(video)

SellerDaoJDBC and DaoFactory

(video)

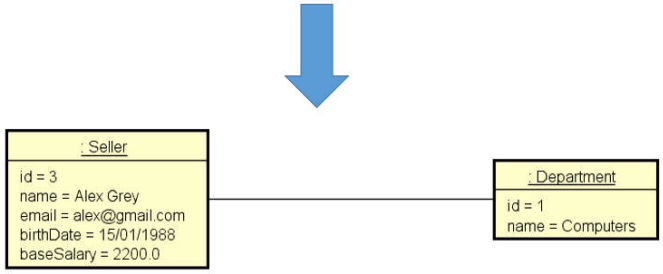
findById implementation

SQL Query:

```
SELECT seller.*,department.Name as DepName
FROM seller INNER JOIN department
ON seller.DepartmentId = department.Id
WHERE seller.Id = ?
```

ResultSet (table)

Id	Name	Email	BirthDate	BaseSalary	DepartmentId	DepName
3	Alex Grey	alex@gmail.com	1988-01-15 00:00:00.000000	2200	1	Computers



associated objects

Reusing instantiation

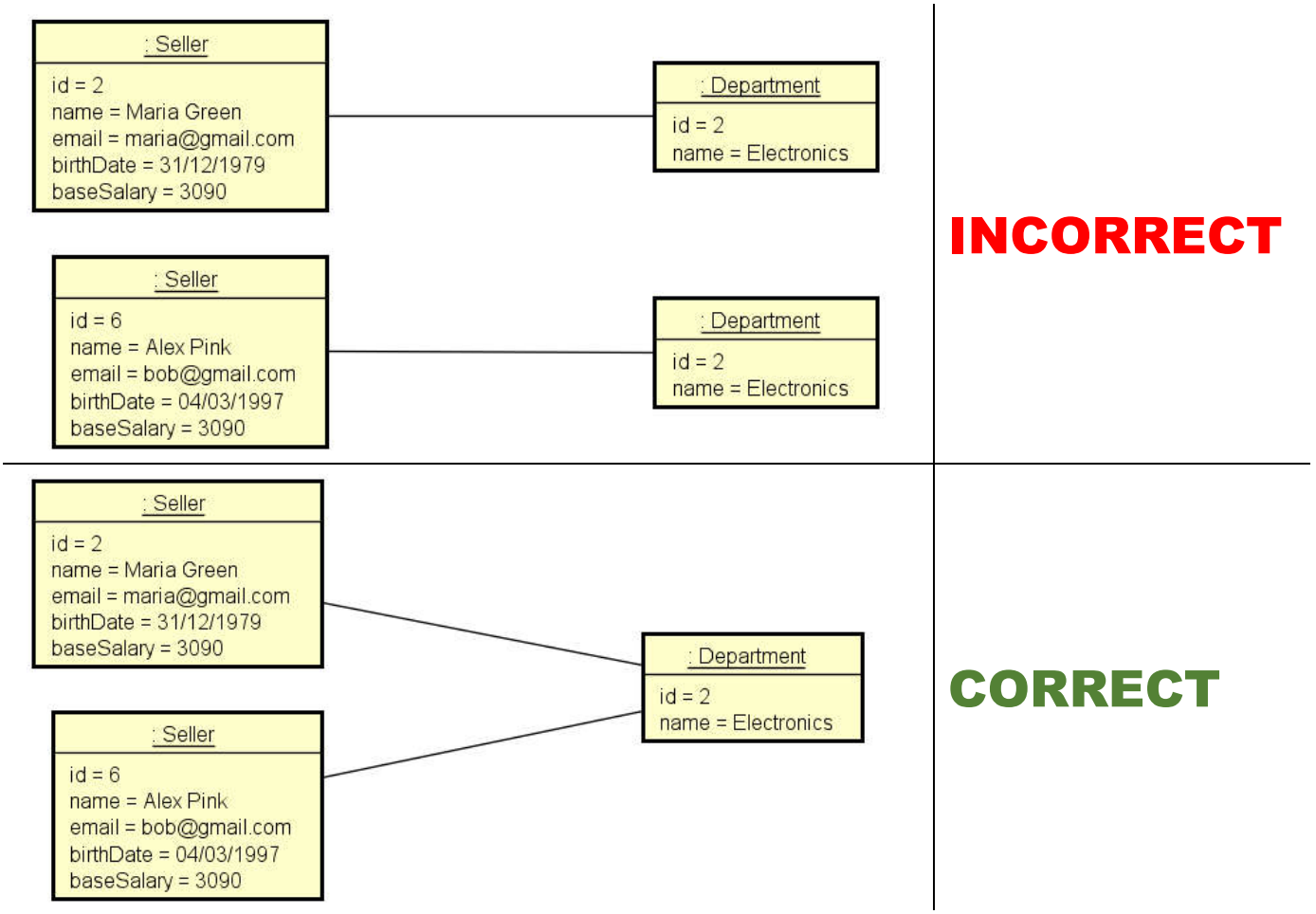
```
private Seller instantiateSeller(ResultSet rs, Department dep) throws SQLException {
	Seller obj = new Seller();
	obj.setId(rs.getInt("Id"));
	obj.setName(rs.getString("Name"));
	obj.setEmail(rs.getString("Email"));
	obj.setBaseSalary(rs.getDouble("BaseSalary"));
	obj.setBirthDate(rs.getDate("BirthDate"));
	obj.setDepartment(dep);
	return obj;
}

private Department instantiateDepartment(ResultSet rs) throws SQLException {
	Department dep = new Department();
	dep.setId(rs.getInt("DepartmentId"));
	dep.setName(rs.getString("DepName"));
	return dep;
}
```

findByDepartment implementation

SQL Query:

```
SELECT seller.*,department.Name as DepName
FROM seller INNER JOIN department
ON seller.DepartmentId = department.Id
WHERE DepartmentId = ?
ORDER BY Name
```



findAll implementation

SQL Query:

```
SELECT seller.*,department.Name as DepName
FROM seller INNER JOIN department
ON seller.DepartmentId = department.Id
ORDER BY Name
```

insert implementation

SQL Query:

```
INSERT INTO seller
(Name, Email, BirthDate, BaseSalary, DepartmentId)
VALUES
(?, ?, ?, ?, ?)
```

update implementation

SQL Query:

```
UPDATE seller
SET Name = ?, Email = ?, BirthDate = ?, BaseSalary = ?, DepartmentId = ?
WHERE Id = ?
```

delete implementation

SQL Query:

```
DELETE FROM seller
WHERE Id = ?
```

DepartmentDao implementation and test

Checklist:

- DepartmentDaoJDBC
- DaoFactory
- Program2

http://github.com/acenelio/demo-dao-jdbc