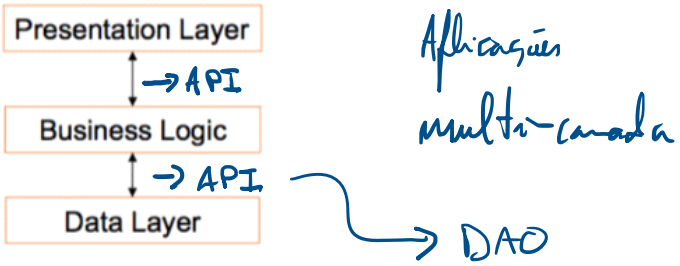
Aula PL #01

21 de setembro de 2023 11:06



Aplicações multi-camada

Figura 1.1: Padrão *Model-Delegate*

MVC (?)

Model View Controller

- Camada de apresentação
- Camada de negócio
- Camada de dados
  - Persistência
    - Ficheiros
    - Base de Dados

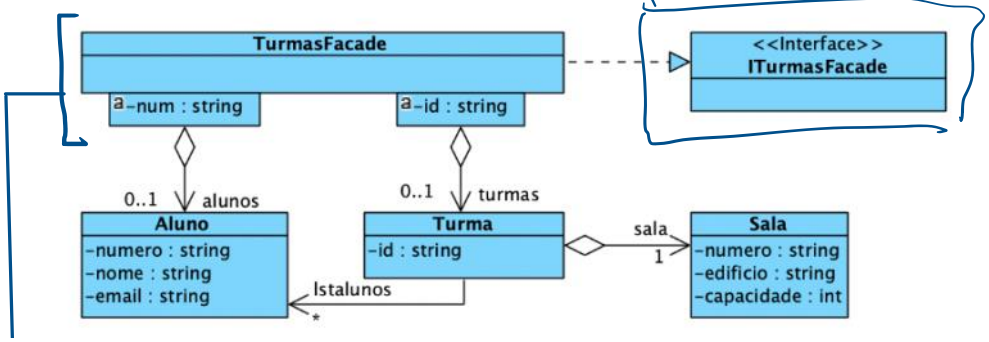
Classe Turma
- Map de Alunos

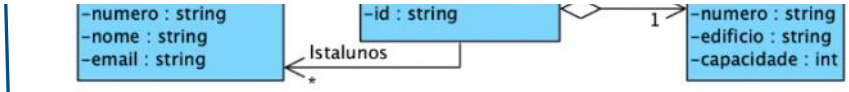
VS

Base de Dados
- Tabela para Turma e Alunos

Deixam de existir estruturas de dados em memória. Passam a existir métodos de conexão com a base de dados, que inserem dados nas tabelas e que constrói os dados em objetos quando necessário.

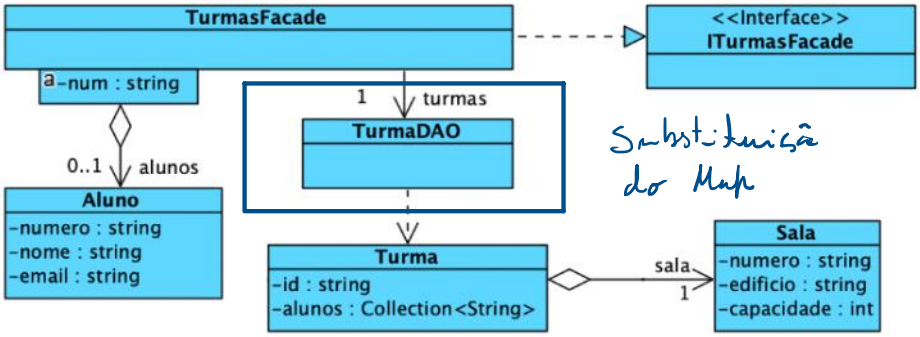
Interface
⇒ Flexibilidade





→ adiciona aluno
altera sala de aluno
...

DAO = Data Access Object



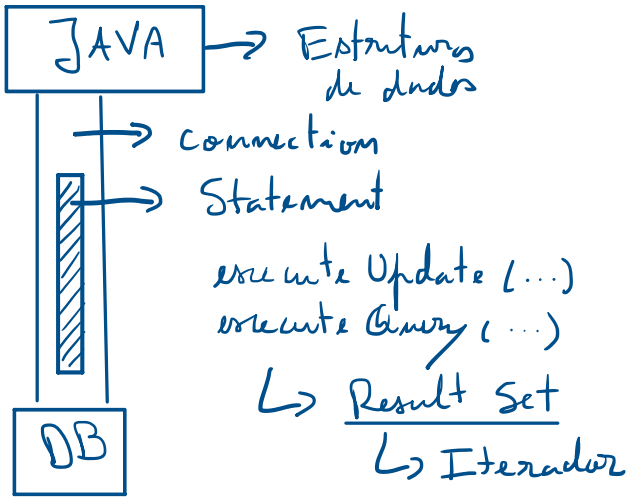
Substituição do Map

Sem alterar os métodos do facade (?)

⇒ TurmaDAO implementa a interface Map

⇒ REESTRUTURAÇÃO da aplicação

jdbc → java database connectivity



↳ Result Set
↳ Iterador

- Criação de tabelas

```
1 usage
private TurmaDAO() {
    try (Connection conn = DriverManager.getConnection(DAOconfig.URL, DAOconfig.USERNAME, DAOconfig.PASSWORD);
         Statement stm = conn.createStatement()) {
        String sql = "CREATE TABLE IF NOT EXISTS salas (" +
                "Num varchar(10) NOT NULL PRIMARY KEY," +
                "Edificio varchar(45) DEFAULT NULL," +
                "Capacidade int(4) DEFAULT 0)";
        stm.executeUpdate(sql);
```

privatestaticTurmaDAO*singleton*=null;

This keyword means that the singleton variable is associated with the class itself, rather than with instances (objects) of the class. In other words, there is only one singleton variable shared among all instances of the class, and it can also be accessed using the class name, e.g., TurmaDAO.singleton.

```
(Connection conn = DriverManager.getConnection(DAOconfig.URL, DAOconfig.USERNAME, DAOconfig.PASSWORD)
 Statement stm = conn.createStatement();
 ResultSet rs = stm.executeQuery( sql: "SELECT count(*) FROM alunos")) {
```

Connection ⇒ Statement (1) ⇒ Result Set (2)

(1)

The object used for executing a static SQL statement and returning the results it produces.
By default, only one `ResultSet` object per `Statement` object can be open at the same time. Therefore, if the reading of one `ResultSet` object is interleaved with the reading of another, each must have been generated by different `Statement` objects. All execution methods in the `Statement` interface implicitly close a current `ResultSet` object of the statement if an open one exists.

(2)

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The `next` method moves the cursor to the next row, and because it returns `false` when there are no more rows in the `ResultSet` object, it can be used in a `while` loop to iterate through the result set.
A default `ResultSet` object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce `ResultSet` objects that are scrollable and/or updatable. The following code fragment, in which `con` is a valid `Connection` object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See `ResultSet` fields for other options.

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The `next` method moves the cursor to the next row, and because it returns `false` when there are no more rows in the `ResultSet` object, it can be used in a `while` loop to iterate through the result set.

A default `ResultSet` object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce `ResultSet` objects that are scrollable and/or updatable. The following code fragment, in which `con` is a valid `Connection` object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See `ResultSet` fields for other options.

```
        Statement stmt = con.createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
        // rs will be scrollable, will not show changes made by others,
        // and will be updatable
```

The `ResultSet` interface provides *getter* methods (`getBoolean`, `getLong`, and so on) for retrieving column values from the current row. Values can be retrieved using either the index number of the column or the name of the column. In general, using the column index will be more efficient. Columns are numbered from 1. For maximum portability, result set columns within each row should be read in left-to-right order, and each column should be read only once.

For the getter methods, a JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value. The JDBC specification has a table showing the allowable mappings from SQL types to Java types that can be used by the `ResultSet` getter methods.

Column names used as input to getter methods are case insensitive. When a getter method is called with a column name and several columns have the same name, the value of the first matching column will be returned. The column name option is designed to be used when column names are used in the SQL query that generated the result set. For columns that are NOT explicitly named in the query, it is best to use column numbers. If column names are used, the programmer should take care to guarantee that they uniquely refer to the intended columns, which can be assured with the SQL *AS* clause.

A set of updater methods were added to this interface in the JDBC 2.0 API (Java™ 2 SDK, Standard Edition, version 1.2). The comments regarding parameters to the getter methods also apply to parameters to the updater methods.

The updater methods may be used in two ways:

1. to update a column value in the current row. In a scrollable ResultSet object, the cursor can be moved backwards and forwards, to an absolute position, or to a position relative to the current row. The following code fragment updates the NAME column in the fifth row of the ResultSet object rs and then uses the method updateRow to update the data source table from which rs was derived.

```
rs.absolute(5); // moves the cursor to the fifth row of rs
rs.updateString("NAME", "AINSWORTH"); // updates the
   // NAME column of row 5 to be AINSWORTH
rs.updateRow(); // updates the row in the data source
```

2. to insert column values into the insert row. An updatable ResultSet object has a special row associated with it that serves as a staging area for building a row to be inserted. The following code fragment moves the cursor to the insert row, builds a three-column row, and inserts it into rs and into the data source table using the method insertRow.

```
rs.moveToInsertRow(); // moves cursor to the insert row
rs.updateString(1, "AINSWORTH"); // updates the
   // first column of the insert row to be AINSWORTH
rs.updateInt(2,35); // updates the second column to be 35
rs.updateBoolean(3, true); // updates the third column to true
rs.insertRow();
rs.moveToCurrentRow();
```

A ResultSet object is automatically closed when the Statement object that generated it is closed, re-executed, or used to retrieve the next result from a sequence of multiple results.
The number, types and properties of a ResultSet object's columns are provided by the ResultSetMetaData object returned by the ResultSet.getMetaData method.

- Reconstrução de um objeto da classe "Turma"

```
5 usages
private String id;
6 usages
private Sala sala;
7 usages
private final Collection<String> lstalunos;
```

(2) (1)

(1)

```
// Reconstruir a colecção de alunos da turma
Collection<String> alunos = getAlunosTurma(key.toString(), stm);
```

```
Collection<String> alunos = new TreeSet<>();
try (ResultSet rsa = stm.executeQuery( sql: "SELECT Num FROM alunos WHERE Turma='"+tid+"'")) {
    while(rsa.next()) {
        alunos.add(rsa.getString( columnLabel: "Num"));
    }
```

Adicionar o número de cada aluno dessa turma à collection.

(2)

```java
Sala s = null;
String sql = "SELECT * FROM salas WHERE Num='"+rs.getString( columnLabel: "Sala")+"'";
try (ResultSet rsa = stm.executeQuery(sql)) {
    if (rsa.next()) {  // Encontrou a sala
        s = new Sala(rs.getString( columnLabel: "Sala"),
                     rsa.getString( columnLabel: "Edificio"),
                     rsa.getInt( columnLabel: "Capacidade"));
    } else {
        // BD inconsistente!! Sala não existe - tratar com excepções.
    } // catch é feito no try inicial - este try serve para fechar o ResultSet automaticamente
      // Nota: abrir um novo ResultSet no mesmo Statement fecha o ResultSet anterior
}
```

A tabela "Turma" tem como chave estrangeira: "Sala"

⇒

```java
t = new Turma(rs.getString( columnLabel: "Id"), s, alunos);
```

A reconstrução de um objeto aluno é mais simples

```java
5 usages
private String numero;
4 usages
private String nome;
4 usages
private String email;
```

⇒

```java
if (rs.next()) {
    a = new Aluno(
            rs.getString( columnLabel: "Num"),
            rs.getString( columnLabel: "Nome"),
            rs.getString( columnLabel: "Email")
    );
```

Exercício 2:

```java
4 usages
private Map<String, Aluno> mapAlunos;

2 usages
public Turma() {
    this.id = "";
    this.sala = new Sala();
    this.lstalunos = new TreeSet<>();
    this.mapAlunos = AlunoDAO.getInstance();
}
```

Novo MAP na classe TURMA

Instância única usada

```
}
```

Instância única usada pelo código
↳ SINGLETON

```
public Collection<Aluno> getObjAlunos() {
    return this.mapAlunos.values();
}
```

Exercício 3

- Salas DAO
- Operações sobre salas - Adicionar
  - Listar
  - Consultar
- Apenas associar salas registadas às turmas