

Medii de proiectare și programare

2024-2025

Curs 3

Conținut curs 3

- Accesul la baze de date relaționale (Java, configurare)
- Ierarhia repository
- ADO.NET
 - Configurarea(C# app.config)
- Adnotări

Clasa Connection

- Reprezintă o sesiune cu o bază de date specifică.
- Orice instrucțiune SQL este executată și rezultatele sunt transmise folosind contextul unei conexiuni.
- Metode:
 - `close()` , `isClosed():boolean`
 - `createStatement():Statement` //overloaded
 - `prepareCall():CallableStatement` //overloaded
 - `prepareStatement():PreparedStatement` //overloaded
 - `rollback()`
 - `setAutoCommit(boolean)` //tranzactii
 - `getAutoCommit():boolean`
 - `commit()`

Clasa Statement

- Se folosește pentru executarea unei instrucțiuni SQL și pentru transmiterea rezultatului.
- Metode:
 - `execute(sql:String, ...):boolean` //pentru orice instructiune SQL
 - `getResultSet():ResultSet`
 - `getUpdateCount():int`
 - `executeQuery(sql:String, ...):ResultSet` //pentru SELECT
 - `executeUpdate(sql:String, ...):int` //INSERT,UPDATE,DELETE
 - `cancel()`
 - `close()`

Exemplu Statement – Structura bazei de date



	Field Name	Data Type
🔑	ID	AutoNumber
	title	Text
	authors	Text
	isbn	Text
	year	Number
▶		

Statement exemplu

```
//Conectarea la o baza de date SQLite
Connection conn=DriverManager.getConnection("jdbc:sqlite:/Users/teste/database/
books.db");
//select
try(Statement stmt=conn.createStatement()){
    try(ResultSet rs=stmt.executeQuery("select * from books")){
        ...
    }
}catch(SQLException ex){
    System.err.println(ex.getSQLState());
    System.err.println(ex.getErrorCode());
    System.err.println(ex.getMessage());
}
//update
String upString="update books set isbn='tj234' where isbn='tj237'"
try(Statement stmt=conn.createStatement()){
    int res=stmt.executeUpdate(upString);
    if (res==0) {...} //nu s-a modificat nicio inregistrare
}catch(SQLException ex){...}
```

Statement exemplu

//insert

```
String insert="insert into books (title, authors, isbn, year) values  
('Nuvele', 'Mihai Eminescu', '4567567', 2008)";  
try(Statement stmt=conn.createStatement()) {  
    int res=stmt.executeUpdate(insert);  
    if (res==0){//nu s-a inserat  
    }  
} catch (SQLException e) {  
    System.out.println("Insert error "+e);  
}
```

//delete

```
String delString="delete from books where isbn='tj234'"  
try(Statement stmt=conn.createStatement()){  
    int res=stmt.executeUpdate(delString);  
    if (res==0){} //nu s-a sters nicio inregistrare  
}catch(SQLException ex){  
    //...  
}
```

ResultSet

- Conține o tabelă ce reprezintă rezultatul unei instrucțiuni SELECT.
- Un obiect de tip `ResultSet` conține un cursor care indică linia curentă din tabelă.
- La început cursorul este poziționat înaintea primei linii din tabelă.
- Metoda **`next`** mută cursorul pe următoarea linie din tabelă. Rezultatul returnat este **`false`**, dacă nu mai există linii neparcurse în obiectul `ResultSet`.
- Metoda **`next`** se folosește pentru a parcurge toate liniile din tabelă.
- Se pot configura anumite proprietăți (daca tabela poate fi modificata, modul de parcurgere, etc.) .
- Configurarea se face în momentul apelului metodei de tip **`createStatement(...)`** :

```
Statement stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = stmt.executeQuery("SELECT name, address FROM users");  
    // rs poate fi iterat, nu va fi notificat de modificari facute  
    //de alti utilizatori ai BD, si poate fi actualizat.
```


ResultSet

- Metode:
 - `absolute(row:int)`
 - `relative(n:int)`
 - `afterLast()`, `beforeFirst()`, `first()`, `last()`, `next():boolean`
 - `getRow():int`
 - `getInt(columnIndex|columnLabel):int`
 - `getFloat(...)`, `getString(...)`, `getObject(...)`, etc
 - `updateInt(columnIndex|columnLabel, newValue)`
 - `updateFloat(...)`, `updateString(...)`, etc
 - `updateRow()`
 - `refreshRow()`
 - `rowDeleted()`, `rowInserted()`, `rowUpdated()`

ResultSet

- Implicit un obiect de tip **ResultSet** este unidirecțional, cu parcurgerea înainte și nu poate fi modificat (actualizat).

```
try(Statement stmt=conn.createStatement()) {  
    try(ResultSet rs=stmt.executeQuery("select * from books")) {  
        while(rs.next()) {  
            System.out.println("Book "+rs.getString("title")+  
                '+rs.getString("author")+ ' '+rs.getInt("year"));  
        }  
    }  
} catch(SQLException ex) {  
  
    // ...  
}
```

Clasa PreparedStatement

- Unui obiect de tip **PreparedStatement** i se transmite instrucțiunea SQL în momentul creării.
- Instrucțiunea SQL este transmisă sistemului de gestiune a bazei de date (SGBD), unde este compilată.
- Când se execută instrucțiunea asociată unui **PreparedStatement**, SGBD execută direct instrucțiunea SQL fără a o reverifica în prealabil.
- Este mai eficientă decât Statement.
- **Poate să aibă parametrii.** Aceștia sunt **marcați folosind ‘?’**.

```
PreparedStatement preStmt = con.prepareStatement(  
    "select * from books WHERE year=?");
```

- Valoarea unui parametru este transmisă folosind metodele de tip **setXYZ**, unde **xyz** reprezinta tipul parametrului.
- **Pozițiile parametrilor încep de la 1.**

```
preStmt.setInt(1, 2008);  
ResultSet rs=preStmt.executeQuery();
```

Tranzacții

- Implicit, fiecare instrucțiune SQL este tratată ca și o tranzacție și este înregistrată/operată imediat după execuție.
- Comportamentul implicit poate fi modificat folosind metoda `setAutoCommit(false)` din clasa `Connection`.
- Metode:
 - `commit`
 - `rollback`
 - `setSavePoint`

Tranzacții - exemplu

```
con.setAutoCommit(false);  
PreparedStatement updateSales = con.prepareStatement(  
    "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");  
updateSales.setInt(1, 50);  
updateSales.setString(2, "Black");  
updateSales.executeUpdate();  
PreparedStatement updateTotal = con.prepareStatement(  
    "UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME  
    LIKE ?");  
updateTotal.setInt(1, 50);  
updateTotal.setString(2, "Black");  
updateTotal.executeUpdate();  
con.commit();  
con.setAutoCommit(true);
```

Proceduri stocate

- O procedură stocată este un grup de instrucțiuni SQL care formează o unitate logica și îndeplinesc o anumită sarcină.
- Ele sunt folosite pentru a îngloba o serie de operațiuni sau interogări ce trebuie executate pe un server de baze de date.
- De exemplu, operațiunile de pe o bază de date angajat (angajarea, concedierea, promovarea, cautarea) ar putea fi codificate ca proceduri stocate executate în funcție de codul cerere.
- Procedurile stocate pot fi compilate și executate cu diferiți parametri și pot avea orice combinație de intrare, ieșire sau intrare/ieșire.

CallableStatement

- Este folosită pentru executarea procedurilor stocate.
- Tehnologia JDBC API furnizează o sintaxă de apelare a procedurilor stocate independentă de SGBD folosit.
- Sintaxa folosită are două variante:
 - conține un parametru de tip rezultat
 - nu conține un parametru de tip rezultat.
- Dacă se folosește prima varianta, parametrul de tip rezultat trebuie să fie înregistrat ca și parametru de tip OUT. Ceilalți parametrii pot fi folosiți pentru intrare, ieșire sau ambele.
- Parametrii sunt referiți secvențial, folosind numere, primul parametru fiind pe poziția 1.

```
{?= call <procedure-name>[ (<arg1>,<arg2>, ...)]}  
{call <procedure-name>[ (<arg1>,<arg2>, ...)]}
```

```
CallableStatement cs = con.prepareCall("{call SHOW_SUPPLIERS}");  
ResultSet rs = cs.executeQuery();
```

Properties

- Clasa **Properties** (pachetul **java.util**) se folosește pentru a păstra perechi cheie-valoare. Perechile pot fi citite sau salvate dintr-un/într-un flux de date (ex. fișier). Cheia și valoarea sunt de tip String, cheia fiind unică.

```
//exemplu.properties  
tasksFile=tasks.txt  
inputDir=input  
outputDir=output
```

```
//Citirea fișierului cu proprietăți  
Properties props=new Properties();  
try {  
    props.load(new FileInputStream("exemplu.properties"));  
} catch (IOException e) {  
    System.out.println("Eroare: "+e);  
}
```


Properties

- Metode:
 - `getProperty(cheie:String):String`
 - `setProperty(c:String, v:String):Object`
 - `list(PrintWriter)`
 - `load(Reader)`
 - `store(w:Writer, comentarii:String)`

```
Properties props=new Properties();
try {
    props.load(new FileInputStream("exemplu.properties"));
} catch (IOException e) {
    System.out.println("Eroare: "+e);
}
String tasksFile=props.getProperty("tasksFile");
if (tasksFile==null) //proprietatea nu a fost gasita in fisier
    System.out.println("fisier incorect");
...
```

System + Properties

- Metode din clasa System:
 - setProperty(Properties)
 - setProperty(c:String, v:String):String
 - getProperty(String):String
 - ...

```
Properties serverProps=new Properties(System.getProperties());
try {
    serverProps.load(new FileReader("exemplu.properties"));
    System.setProperty(serverProps);
    System.getProperties().list(System.out);
} catch (IOException e) {
    System.out.println("Eroare "+e);
}
String tasksFile=System.getProperty("tasksFile");
```

Exemplu configurare BD (System)

```
//Fisierul bd.properties sau bd.config  
jdbc.url=jdbc:mysql://localhost/mpp  
jdbc.user=test  
jdbc.pass=test
```

```
//cod
```

```
Connection getConnection() {  
    String url=System.getProperty("jdbc.url");  
    String user=System.getProperty("jdbc.user");  
    String pass=System.getProperty("jdbc.pass");  
    Connection con=null;  
    try {  
        con= DriverManager.getConnection(url,user,pass);  
    } catch (SQLException e) {  
        System.out.println("Eroare stabilire conexiune "+e);  
    }  
    return con;  
}
```

Exemplu configurare BD

```
//Fisierul bd.properties sau bd.config  
jdbc.url=jdbc:mysql://localhost/mpp  
jdbc.user=test  
jdbc.pass=test
```

```
//cod
```

```
Connection getConnection(Properties props) {  
    String url=props.getProperty("jdbc.url");  
    String user=props.getProperty("jdbc.user");  
    String pass=props.getProperty("jdbc.pass");  
    Connection con=null;  
    try {  
        con= DriverManager.getConnection(url,user,pass);  
    } catch (SQLException e) {  
        System.out.println("Eroare stabilire conexiune "+e);  
    }  
    return con;  
}
```

Dependențe driver JDBC - Gradle

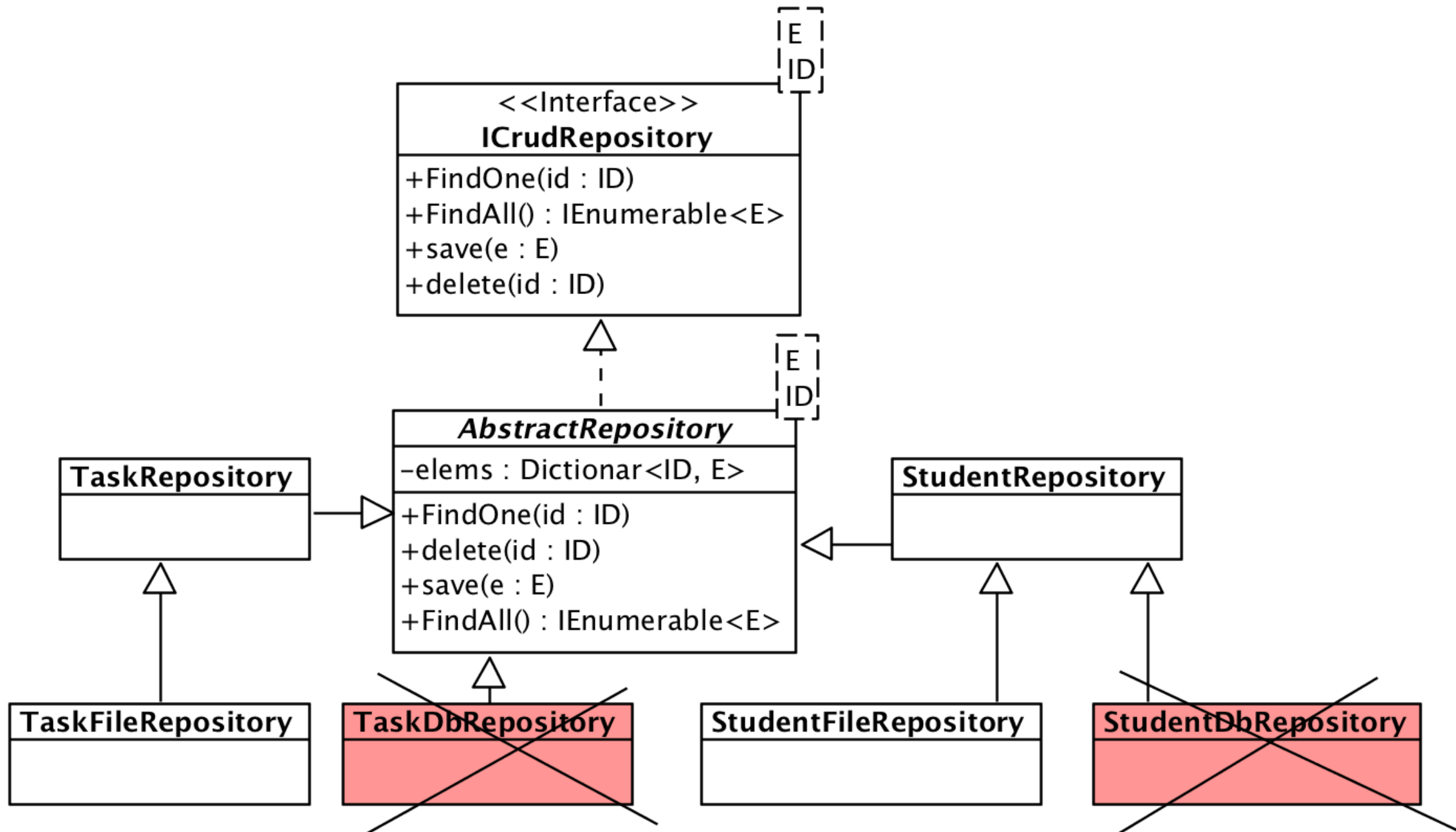
```
//Fisierul build.gradle
dependencies {
    testImplementation group: 'org.junit.jupiter', name: 'junit-jupiter-api', version:
        '5.8.2'
    testImplementation group: 'org.junit.jupiter', name: 'junit-jupiter-engine',
        version: '5.8.2'
    //jurnalizare (cautati pe Maven Repository)
    implementation group: 'org.apache.logging.log4j', name: 'log4j-core', version:
        '2.18.0'
    implementation group: 'org.apache.logging.log4j', name: 'log4j-api', version:
        '2.18.0'
    //drivere conectare la baza de date (cautati pe Maven Repository)
    runtimeOnly group: 'org.xerial', name: 'sqlite-jdbc', version: '3.36.0.3'

    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client:2.1.2'
    runtimeOnly 'mysql:mysql-connector-java:5.1.20'
}

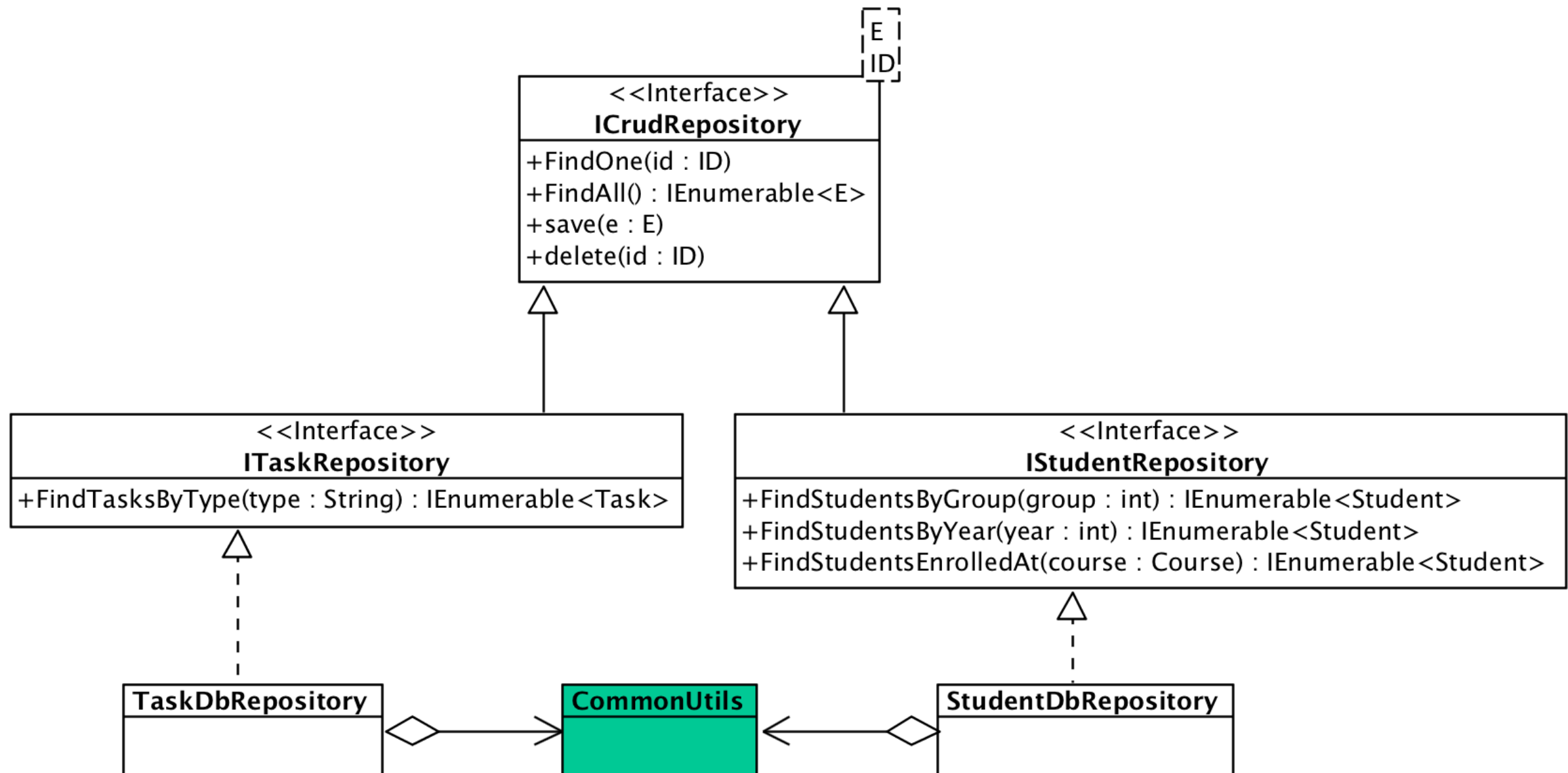
//Maven repository
//https://mvnrepository.com/
```

Exemplu Java

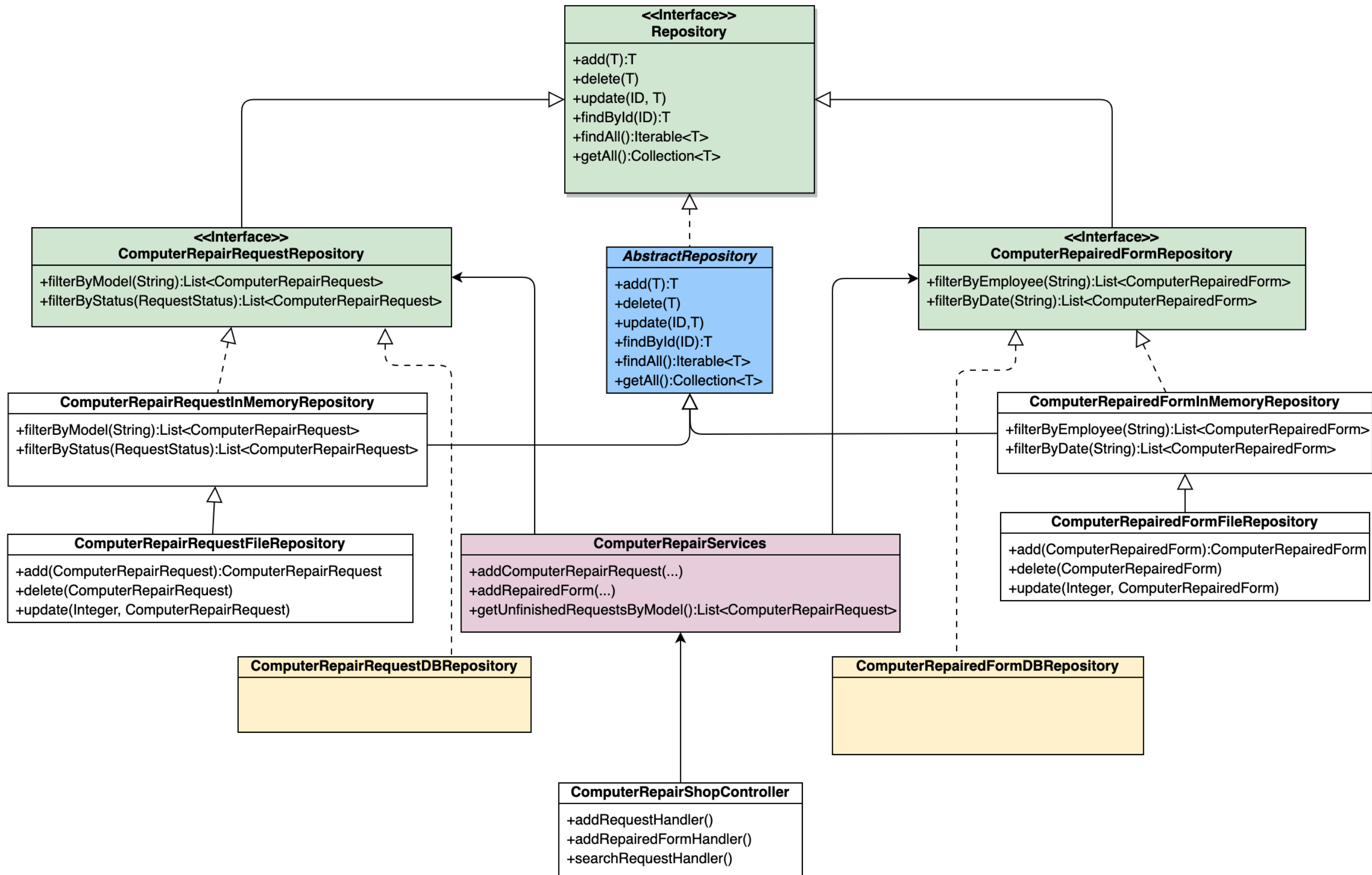
Ierarhie repositories (1)



Ierarhie repositories (2)



Ierarhie repositories (3)



ADO.NET

- ADO.NET este o bibliotecă orientată pe obiecte care permite unei aplicații să interacționeze cu diferite surse de date:
 - baze de date relaționale
 - fișiere text
 - fișiere Excel
 - fișiere XML
- Conține 4 spații de nume pentru interacțiunea cu 4 tipuri de baze de date:
 - SQL Server
 - Oracle
 - Surse ODBC
 - OLEDB.

ADO.NET

Spații de nume

- **System.Data**—Toate clasele generice pentru accesarea datelor.
- **System.Data.Common**—Clase comune sau redefinite de furnizori de date specifici.
- **System.Data.Odbc**—Clasele pentru ODBC
- **System.Data.OleDb**—Clasele pentru OLE DB
- **System.Data.Oracle**—Clasele pentru Oracle
- **System.Data.SqlClient**—Clasele pentru SQL Server
- **System.Data.SqlTypes**—Tipurile de date SQL Server

System.Data

- Conține clasele și interfețele folosite indiferent de sistemul de gestiune a bazelor de date.
- **DataSet**—Clasa pentru lucru offline. Poate conține o mulțime de **DataTables** și relații între acestea.
- **DataTable**—Un container ce conține una sau mai multe coloane. Când este populat va avea una sau mai multe **DataRows** conținând informația.
- **DataRow**—O mulțime de valori corespunzând unei linii dintr-o tabelă dintr-o bază de date relațională, sau unei linii dintr-o foaie de calcul.
- **DataColumn**—Conține definiția unei coloane dintr-o tabelă: numele și tipul.
- **DataRelation**—Reprezintă o relație între două tabele dintr-un **DataSet**. Se folosește pentru a reprezenta relația "cheie străină".
- **Constraint**—Definește constrângeri pentru una sau mai multe **DataColumn** (ex. valori unice).

System.Data.Common

- **DataColumnMapping**—Mapează numele unei coloane dintr-o tabelă din baza de date cu numele unei coloane dintr-un **DataTable**.
- **DataTableMapping**—Mapează numele unei tabele dintr-o bază de date cu un **DataTable** dintr-un **DataSet**.
- **DbCommandBuilder**—Generează automat comenzi pentru a sincroniza modificările dintr-un **DataSet** cu baza de date asociată.

ADO.NET API

- ADO.NET conține clase specifice interacțiunii cu anumite tipuri de baze de date.
- Aceste clase implementează o mulțime de interfețe standard din spațiul de nume System.Data, permițând claselor să fie folosite într-o manieră generică, dacă este necesar.
 - **IDbConnection** folosită pentru conectarea la o baza de date.
 - **IDataAdapter** folosită pentru păstrarea instrucțiunilor *select*, *insert*, *update* și *delete* care sunt apoi folosite pentru popularea unui **DataSet** și pentru actualizarea bazei de date.
 - **IDataReader**: folosit ca și un cititor de date, forward-only.
 - **IDbCommand**: folosit ca și wrapper pentru instrucțiuni SQL sau apeluri de proceduri stocate.
 - **IDbDataParameter**: reprezintă un parametru pentru un obiect de tip Command.
 - **IDbTransaction**: folosit pentru reprezentarea unei tranzacții ca și un obiect.

IDbConnection

- Reprezintă o conexiune deschisă către o sursă de date:
 - `SqlConnection, OleDbConnection, OracleConnection, OdbcConnection`
 - `MySqlConnection, SQLiteConnection (Windows),`
`SQLiteConnection (Mono)`
- Metode:
 - `BeginTransaction`
 - `ChangeDatabase`
 - `Open`
 - `Close`
 - `CreateCommand`
- Proprietăți:
 - `ConnectionString, ConnectionTimeout, Database, State`

SqlConnection

- Conectarea la Sql Server

```
var conn = new SqlConnection(  
    "Data Source=(local);Initial Catalog=Northwind;User Id=test;  
    Password=test");
```

- Conectarea la o bază de date Access folosind OleDb

```
String connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=books.mdb";  
var conn=new OleDbConnection(connectionString);
```

- Conectarea la MySql:

```
String connectionString = "Database=mpp;Data Source=localhost;User id=test;"  
    + "Password=passtest;";  
var conn= new MySqlConnection(connectionString);
```

- Conectarea la Sqlite (folosind Mono.Sqlite - specific Linux/MacOS):

```
String connectionString = "URI=file:/Users/test/database/tasks.db,Version=3";  
var conn= new SQLiteConnection(connectionString);
```

<https://www.connectionstrings.com/>

SqlCommand

- Reprezintă o instrucțiune SQL executată când există o conexiune către sursa de date.
 - `SqlCommand`, `OleDbCommand`, `OracleCommand`, `ODBCCommand`
 - `MySqlCommand`, `SQLiteCommand (Mono)`, `SQLiteCommand (Windows)`
- Metode:
 - `ExecuteReader`, `ExecuteNonQuery`, `ExecuteScalar`
 - `CreateParameter`
 - `Cancel`
- Proprietăți:
 - `CommandText`, `CommandTimeout`, `CommandType`, `Connection`, `Parameters`, etc.
- `CommandType`:
 - `Text` (o comandă SQL), `StoredProcedure`, `TableDirect` (numele unei tabele, doar pentru furnizori OleDb).

SqlCommand

- Text:

```
String select = "SELECT ContactName FROM Customers";
```

```
SqlCommand cmd = new SqlCommand(select , conn);
```

- Stored Procedure

```
SqlCommand cmd = new SqlCommand("CustOrderHist", conn);
```

```
cmd.CommandType = CommandType.StoredProcedure;
```

- Table Direct

```
OleDbCommand cmd = new OleDbCommand("Categories", conn);
```

```
cmd.CommandType = CommandType.TableDirect;
```

SqlCommand

- ExecuteNonQuery:

```
string source =...;

string sqlCom = "UPDATE Customers SET ContactName = 'Bob' " +
                "WHERE ContactName = 'Bill'";

using(var conn = new OleDbConnection(source)) {
    conn.Open();

    var cmd = new OleDbCommand(sqlCom, conn);

    int rowsReturned = cmd.ExecuteNonQuery();

    Console.WriteLine("{0} rows affected.", rowsReturned);
}
```

IDbCommand

- `ExecuteReader`:

[illegible]

SqlCommand

- ExecuteScalar:

```
string source = ...;

string select = "SELECT COUNT(*) FROM Customers";

using(var conn = new SqlConnection(source)) {

    conn.Open();

    using(var cmd = new SqlCommand(select, conn)) {

        object o = cmd.ExecuteScalar();

        Console.WriteLine ("Customers: {0}", o);

    }

}
```

IDataReader

- Oferă posibilitatea citirii unui flux sau mai multor fluxuri secvențial (forward-only) obținute prin executarea unei comenzi asupra unei surse de date.
 - `SqlDataReader`, `OleDbDataReader`, `OracleDataReader`, `ODBCDataReader`
 - `MySqlDataReader`, `SQLiteDataReader (Mono)`, `SQLiteDataReader (Windows)`
- O instanță de tip `IDataReader` este obținută apelând metoda `IDbCommand.ExecuteReader`.
- Metode:
 - `Read`
 - `GetBoolean`, `GetByte`, `GetDouble`, `GetFloat`, `GetInt16`, `GetString`, etc.
 - `Close`
- Proprietăți:
 - `Item` (index sau nume), `IsClosed`

IDataReader

```
string source = ...;

string selectCmd = "SELECT name,address FROM persons";

using(var conn = new SqlConnection(source)) {

    conn.Open();

    using(var cmd = conn.CreateCommand()) {

        cmd.CommandText=selectCmd;

        using(var reader = cmd.ExecuteReader()) {

            while(reader.Read())

                Console.WriteLine("{0} {1}", reader["name"] , reader["address"]);

        }

    }

}
```

IDataAdapter

- Reprezintă un set de proprietăți folosite pentru completarea unui DataSet și pentru actualizarea unei surse de date.
 - `SqlDataAdapter`, `OleDbDataAdapter`, `OracleDataAdapter`, `ODBCDataAdapter`
 - `MySqlDataAdapter`, `SqliteDataAdapter(Mono)`, `SQLiteDataAdapter (Windows)`
- Este folosit în asociere cu un DataSet.
- Un `DataSet` este un obiect în memorie care poate păstra mai multe tabele.
- `DataSets` păstrează doar informația, nu interacționează cu sursa de date.
- `IDataAdapter` gestionează conexiunile către sursa de date.
- `IDataAdapter` deschide o conexiune doar când este necesar și o închide imediat ce sarcina și-a încheiat execuția.

IDataAdapter

- Execută următoarele când **populează un DataSet** cu date:
 - Deschide o conexiune la sursa de date
 - Obține și încarcă datele în **DataSet**
 - Închide conexiunea
- Execută următoarele când **actualizează sursa de date** cu modificările din DataSet:
 - Deschide conexiunea
 - Scrie modificările din DataSet în sursa de date.
 - Închide conexiunea
- **Între populare și actualizare conexiunile către sursa de date sunt închise.**
- Metode:
 - **Fill** (adaugă sau actualizează linii în DataSet potrivite cu cele din sursa de date),
 - **Update** (apelează instrucțiunile INSERT, UPDATE, sau DELETE corespunzătoare fiecărei inserări, actualizări sau ștergeri din DataSet)
- Proprietăți: **DeleteCommand**, **InsertCommand**, **SelectCommand**, **UpdateCommand**

IDataAdapter

```
string source =...;

var Connection conn = new MySqlConnection(source);

string select = "SELECT * FROM books";

DataSet data=new DataSet();

var dataAdapter=new MySqlAdapter(select, conn);

dataAdapter.Fill(data, "Books");

DataRowCollection dra=data.Tables["Books"].Rows;

foreach(DataRow in dra)

    Console.WriteLine(dr["isbn"]+dr["author"]+dr["title"]);
```

IDataParameter

- Reprezintă parametrul unui obiect de tip `Command`.
 - `SqlParameter`, `OracleParameter`, `OleDbParameter`, `OdbcParameter`
 - `MySqlParameter`, `SQLiteParameter(Mono)`, `SQLiteParameter (Windows)`
- Membrii
 - `Value`
 - `ParameterName`
 - `DbType`
- `DbType`:
 - `Boolean`, `Date`, `Double`, `Int32`, `String`, etc.

IDataParameter

```
string source = ...;

string select = "SELECT * FROM Customers where city=@City";

using(var conn = new SqlConnection(source)) {

    conn.Open();

    using(var cmd = new SqlCommand(select, conn)) {

        var param = cmd.CreateParameter();

        param.ParameterName = "@City";

        param.Value          ="ABC";

        cmd.Parameters.Add(param);

        using(var reader = cmd.ExecuteReader()) {

            while(reader.Read())

            {

                Console.WriteLine("Contact:{0} Company:{1}", reader["CompanyName"],
                    reader["ContactName"]);

            }

        }

    }

}
```

app.config

- Fișier de configurare pentru aplicații .NET

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>
  <connectionStrings>
    <add name="tasksDB"
          connectionString="URI=file:/Users/xyz/MPP/database/tasks.db,Version=3" />
    <!--
  <add name="tasksDB"
        connectionString="Database=mpp;Data Source=localhost;User id=test;Password=passtest;" />
  -->
</connectionStrings>
</configuration>
```

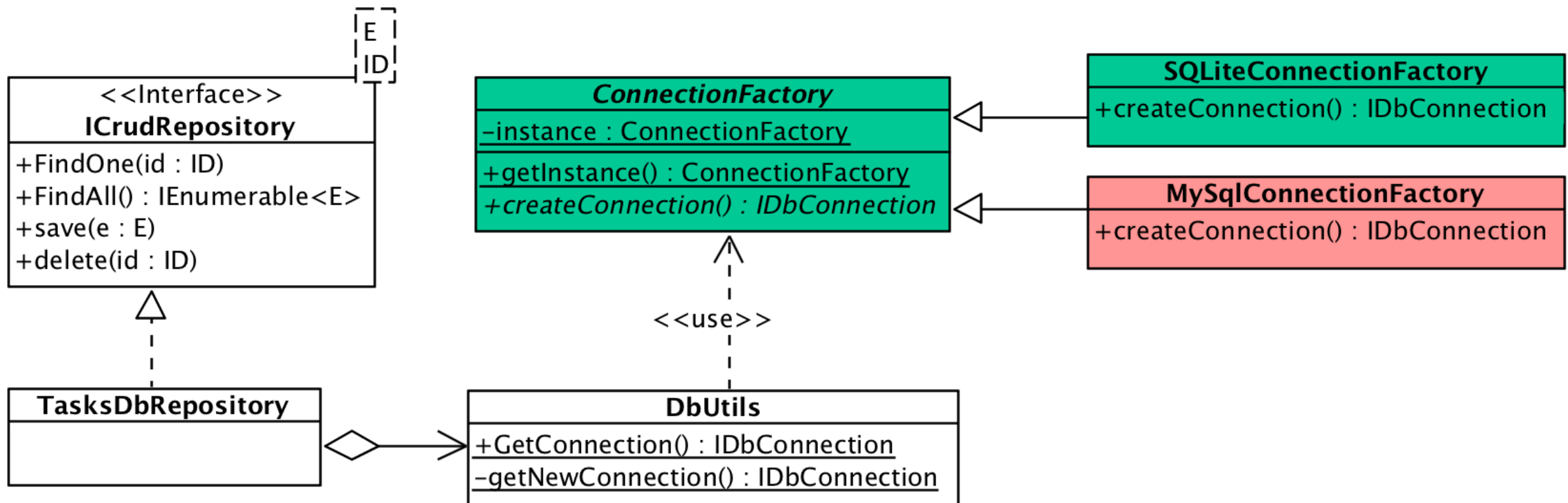
- La compilare fișierul este copiat în directul bin/debug cu numele `NumeApp.exe.config` (unde NumeApp este numele proiectului)

app.config

- Obținerea datelor din app.config
- Clasa `ConfigurationManager` (spațiul de nume `System.Configuration`)

```
static string GetConnectionStringByName(string name){  
    // Presupunem ca nu exista.  
    string returnValue = null;  
  
    // Cauta numele in sectiunea connectionStrings.  
    ConnectionStringSettings settings = ConfigurationManager.ConnectionStrings[name];  
  
    // Daca este gasit, returneaza valoarea asociata la connection string.  
    if (settings != null)  
        returnValue = settings.ConnectionString;  
  
    return returnValue;  
}
```

Arhitectura C#



Exemplu C#

Jurnalizare C#

- Log4net
- Instalare pachete **NuGet**
- Configurare:

<https://stackify.com/log4net-guide-dotnet-logging/>