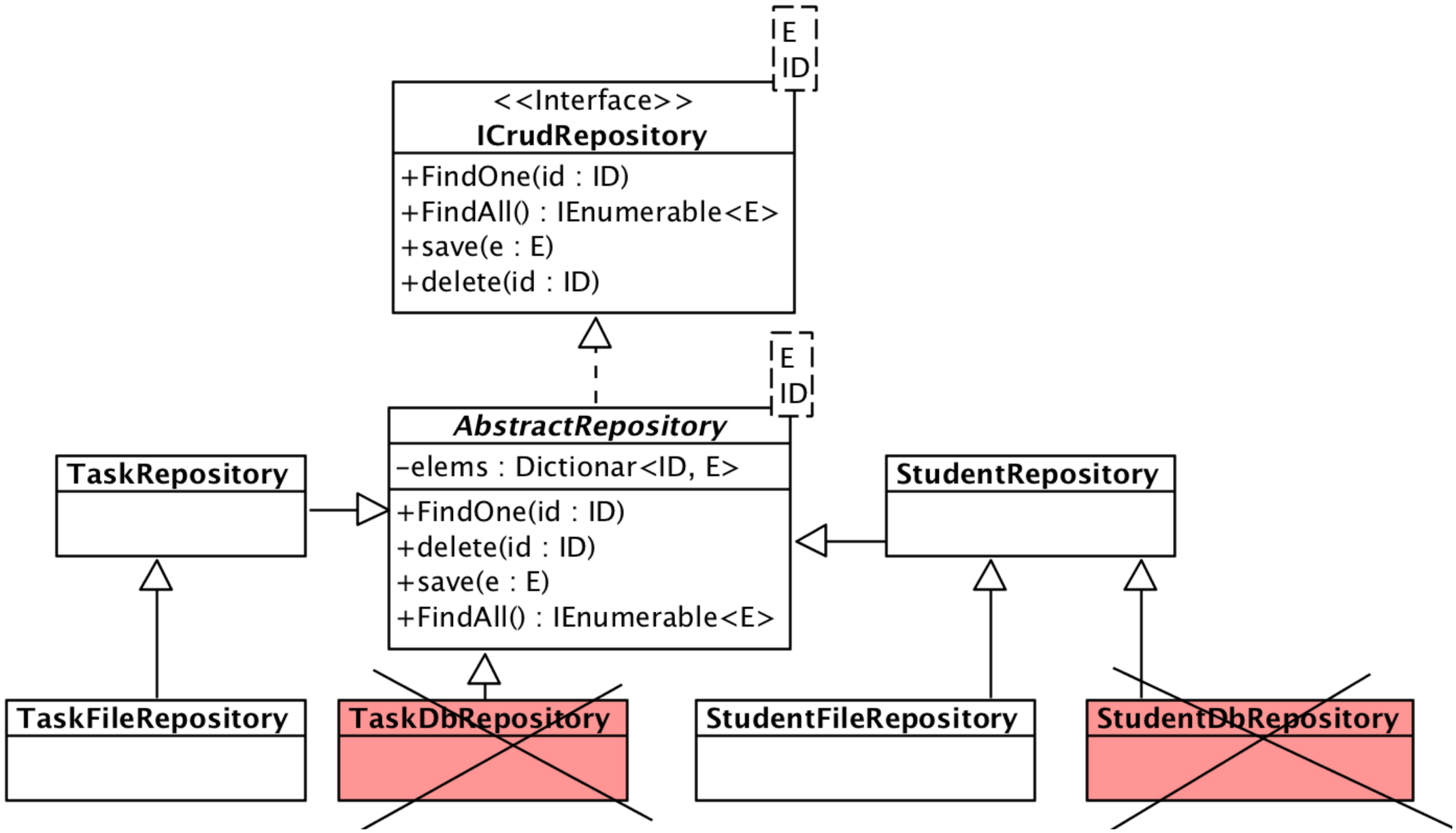
Medii de proiectare și programare

2020-2021
Curs 3

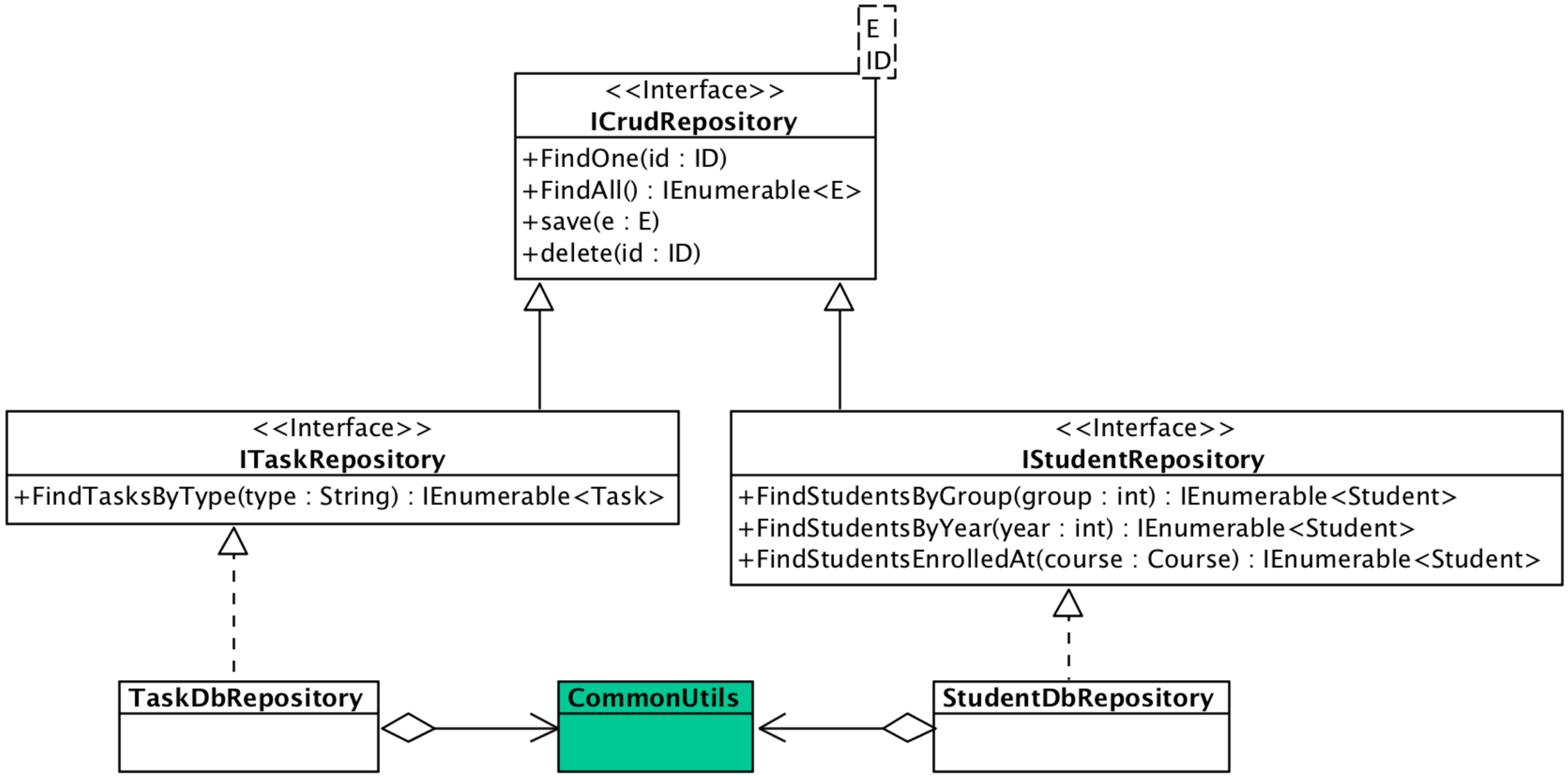
Conținut curs 3

- ADO.NET (cont. curs 2)
- Adnotări
- Java Beans
- Introducere în Spring

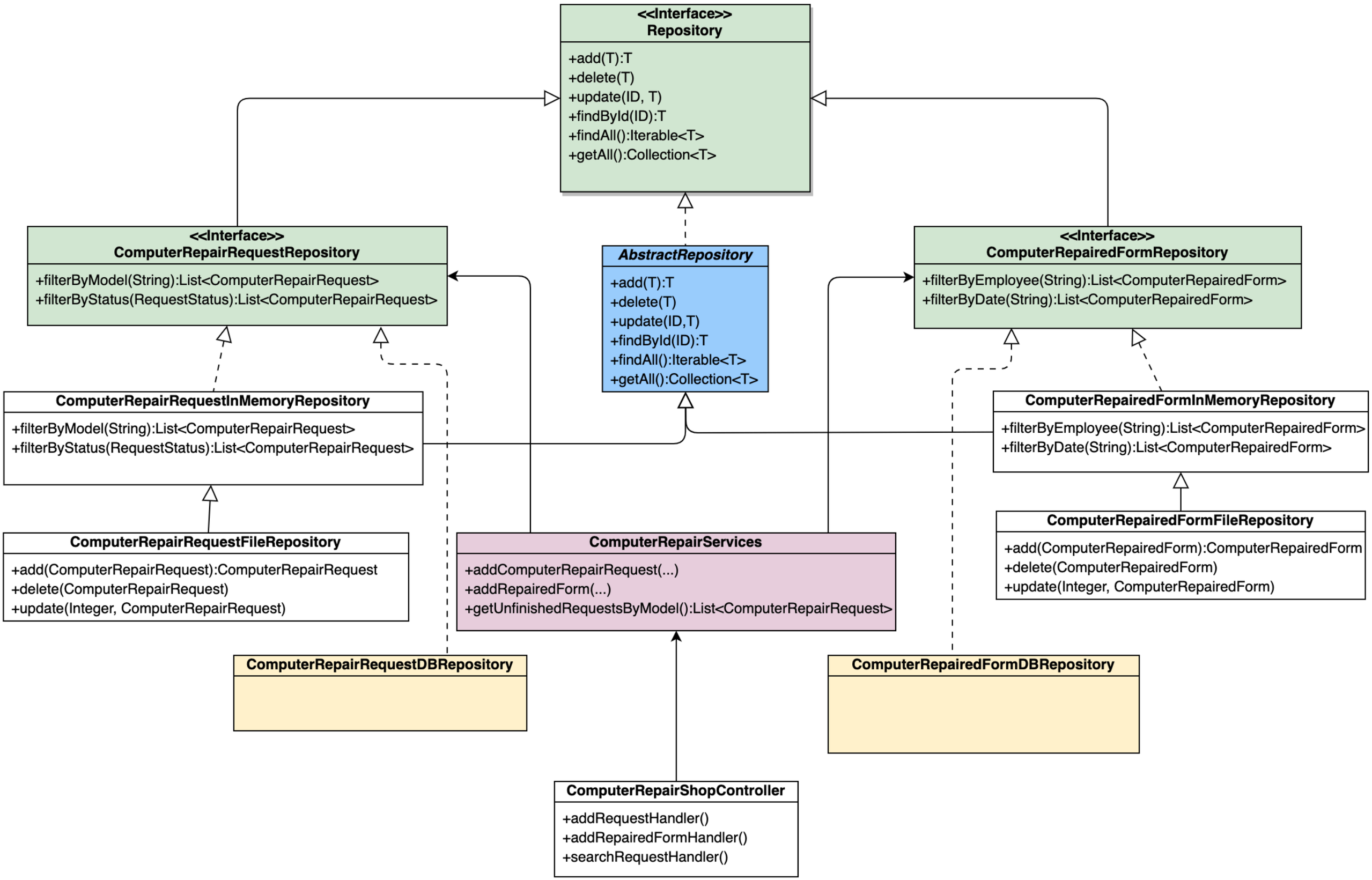
Ierarhie repositories (1)



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`**—Genereaza 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`**, **`SqliteConnection(Mono)`**
- Metode:
  - **`BeginTransaction`**
  - **`ChangeDatabase`**
  - **`Open`**
  - **`Close`**
  - **`CreateCommand`**
- Proprietăți:
  - **`ConnectionString`**, **`ConnectionTimeout`**, **`Database`**, **`State`**

IDbConnection

- 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):

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

IDbCommand

- Reprezintă o instrucțiune SQL executată când există o conexiune către sursa de date.
  - **`SqlCommand, OleDbCommand, OracleCommand, ODBCCommand`**
  - **`MySqlCommand, SqliteCommand (Mono), SQLiteCommand`**
- 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).

IDbCommand

- Text:

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

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

- Stored Procedure

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

cmd.CommandType = CommandType.StoredProcedure;
```

- Table Direct

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

cmd.CommandType = CommandType.TableDirect;
```

IDbCommand

- 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:

```
string source = …;
string select = "SELECT ContactName,CompanyName FROM Customers";
using(var conn = new MySqlConnection(source)){
    conn.Open();
    var cmd = new MySqlCommand(select, conn);
    using(var reader = cmd.ExecuteReader()){
        while(reader.Read())
        {
            Console.WriteLine("Contact:{0} Company:{1}", reader[0] ,
            reader[1]);
        }
     }
}
```

IDbCommand

- ExecuteScalar:

```
string source = …;

string select = “SELECT COUNT(*) FROM Customers”;

using(var conn = new SqliteConnection(source)){
    conn.Open();
    using(var cmd = new SqliteCommand(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`**
- 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 SqliteConnection(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`**
- 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, or 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`**
- 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 SqliteConnection(source)){
     conn.Open();
     using(var cmd = new SqliteCommand(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

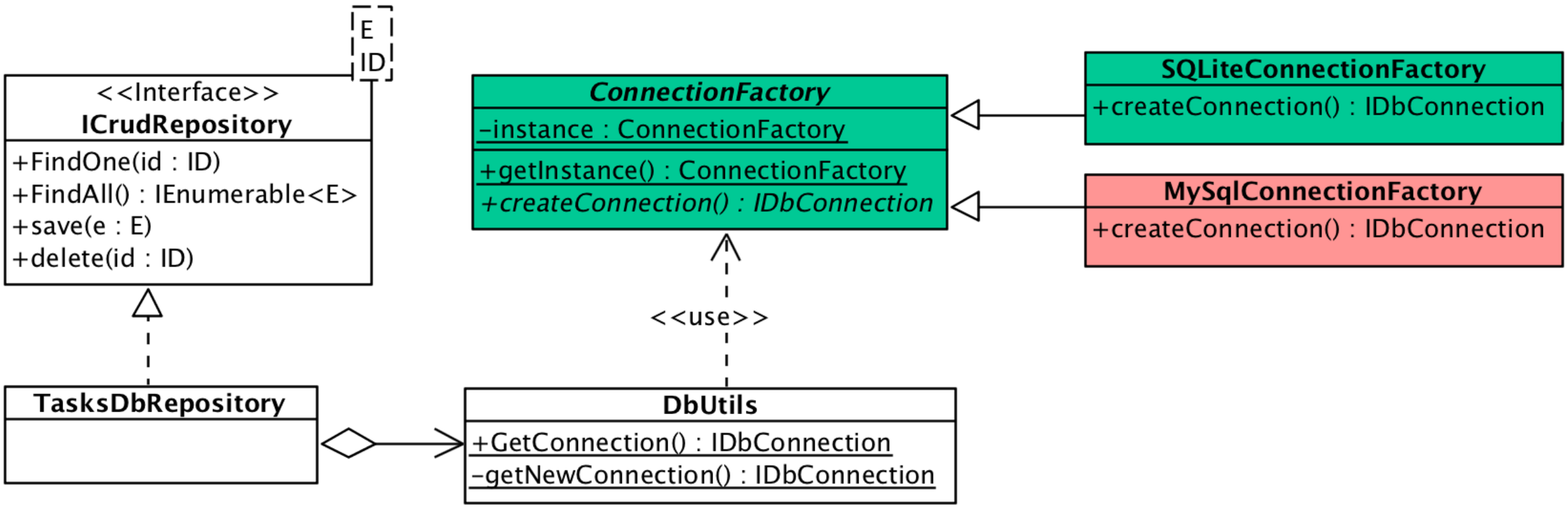
- Obținerea datelor din app.config
  - Clasa ConfigurationManager (spațiul de nume System.Configuration)

```
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#



Adnotări (*Annotations*)

- Începând cu versiunea 1.5
- Adaugă informații unei părți de cod (clasă, metodă, pachet), dar nu fac parte din program. Adnotările nu au nici un efect direct asupra codului pe care îl marchează.
- Utilizări:
  - A furniza informații suplimentare compilatorului. Adnotările pot fi folosite de compilator pentru a detecta erori sau pentru a elimina atenționări.
  - Procesare automata din timpul compilării sau deploymentului. Instrumente soft specializate pot folosi adnotarile pentru a genera automat cod, fișiere XML, etc.
  - Procesare în timpul execuției. Unele adnotări sunt disponibile pentru a fi examinate în timpul execuției codului.

Definirea adnotărilor

```
[declaratii meta-adnotari]
public @interface NumeAdnotare {
   [declaratii elemente]
}
```

Meta-adnotările (pachetul `java.lang.annotation`)(adnotări pentru adnotări) pot fi:

- `@Target(ElementType):` specifică locul din codul sursă unde poate fi folosită adnotarea.
- `CONSTRUCTOR:` declararea unui constructor
- `FIELD:` declararea unui atribut (inclusiv constante enum)
- `LOCAL_VARIABLE:` declararea unei variabile locale
- `METHOD:` declararea unei metode
- `PACKAGE:` declararea unui pachet
- `PARAMETER:` declararea unui parametru
- `TYPE:` declararea unei noi clase, interfețe, adnotări sau enum.

Definirea adnotarilor

Meta-adnotările pot fi:

- `@Retention(RetentionPolicy):` specifică cât timp va fi păstrată adnotarea:
  - `SOURCE:` Adnotările nu sunt salvate la compilare.
  - `CLASS:` Adnotările sunt disponibile în fișierul .class, dar pot fi eliminate de mașina virtuală.
  - `RUNTIME:` Adnotările sunt păstrate de mașina virtuală în timpul execuției și pot fi citite folosind reflecție.
- `@Documented:` Adnotarea este inclusă în documentația Javadocs.
- `@Inherited:` Permit subclaselor să moștenească adnotările părinților.

Elementele unei adnotări

- Sintaxa:

```
Tip numeElement() [default valoare_implicita];
```

unde **`Tip`** poate fi:

  - orice tip primitiv (**`int`**, **`float`**, **`double`**, **`byte`**, etc.)
  - **`String`**
  - **`Class`**
  - Enumerări (**`enum`**)
  - Adnotări (**`annotation`**)
  - Tablouri de tipurile menționate mai sus.

Observații:

1. Dacă se folosește alt tip la declararea unui element, compilatorul va genera eroare.
2. Dacă o adnotare nu conține nici un element, adnotarea se numește de tip *marker*.

Constrângeri valori implicite

- Exista două constrângeri pentru valoarea unui element:

1. Nici un element nu poate avea o valoare nespecificată (fie se declară o valoare implicită, fie se atribuie o valoare pentru fiecare element în momentul folosirii adnotării).
2. Pentru elementele care nu sunt de tip primitiv, nu se acceptă valoarea **`null`** (în momentul folosirii sau ca și valoare implicită).

Adnotări - exemplu

```
import java.lang.annotation.*;


@Target(ElementType.CLASS)
@Retention(RetentionPolicy.RUNTIME)
public @interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   String[] reviewers();
}
```

Folosirea adnotărilor

Adnotarea apare prima, de obicei pe linie proprie, și poate conține elemente.

Observații:

1. Dacă adnotarea conține un singur element numit **`value`**, numele acestuia poate fi omis.
2. Dacă adnotarea nu conține nici un element, parantezele pot fi omise.

```
@ClassPreamble (
   author = “Popescu Vasile",
   date = "3/17/2008",
   currentRevision = 4,
   lastModified = "4/11/2011",
   lastModifiedBy = “Ionescu Matei"
   reviewers = {“Vasilescu Ana", “Marinescu Ion”, “Pop Ioana"}
)
public class A extends B{
//...
}
```

Exemplu adnotări

Declararea:

```
import java.lang.annotation.*;
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface UseCase {
    public int id();
    public String description() default "no description";
}
```

Folosirea

```
public class A{
  @UseCase(id=3, description="abcd")
  public void f(){
  }
}
```

Adnotări standard

- JSE conține 3 adnotari standard:
  - `@Override` pentru a indica că o metoda redefinește o metodă din clasa de baza. Dacă numele metodei sau signatura nu sunt corecte, compilatorul va genera o eroare.

```
class A{
  @Override
  public String toString(){...}
}
```

  - `@Deprecated` pentru a genera o atenționare la compilare când se folosește elementul adnotat (clasă, metodă, etc.)
  - `@SuppressWarnings` spune compilatorului să nu furnizeze anumite atenționări: `unchecked, deprecated`

```
@SuppressWarnings("unchecked", "deprecated")
void metodaA() { }
```

Beans

- Orice clasă Java este un POJO (eng. *Plain Old Java Object*).
- JavaBeans: este o clasă Java specială. Reguli:
  - Trebuie să aibă un constructor implicit (public și fără nici un parametru). Alte instrumente specializate vor folosi acest constructor pentru a instanția un obiect.
  - Atributele trebuie să poată fi accesate folosind metode de tip **`getXyz`**, **`setXyz`** și **`isXyz`** (pentru atribute de tip boolean). Atributele pentru care sunt definite aceste metode se numesc proprietăți, numele proprietății fiind **`xyz`**. Când se modifică sau se dorește valoarea unei proprietăți se apelează una dintre metodele corespunzătoare.
  - Clasa trebuie sa fie serializabilă. Acest lucru permite instrumentelor specializate să salveze și să refacă starea unui JavaBean.
  - Exemplu: Componentele GUI
- Enterprise Java Beans (EJBs): pentru aplicații complexe (tranzacții, securitate, acces la baze de date)

Exemplu Java Beans

```
public class Student implements java.io.Serializable {
    private String nume;
    private int grupa;
    private boolean licentiat;
    private int note[];
    public Student() { }
    public Student(String nume, int grupa, boolean licentiat){...}
    public String getName() { return nume; }
    public void setName(String name) { nume = name;  }
    public int getGrupa() {return grupa;}
    public void setGrupa(int g){grupa=g;}
    public void setLicentiat(boolean l){licentiat=l;}
    public boolean isLicentiat(){ return licentiat;}
    public void setNote(int[] n){ note=n;}
    public int[] getNote(){return note;}
}
```

Introducere în Spring - Motivație

- Orice aplicație medie sau complexă este compusă dintr-o mulțime de obiecte care colaborează pentru atingerea unui scop. Aceste obiecte știu despre celelalte obiecte (asocierile) și comunică prin transmiterea de mesaje.
- Abordarea tradițională pentru crearea asocierilor dintre obiecte (prin instanțiere sau căutare) generează cod complicat care este dificil de reutilizat și testat (folosind unit testing).

//varianta 1

```
class ConcursService{
    private ParticipantiRepositoryMock repo;
  public ConcursService(){
      repo=new ParticipantiRepositoryMock();
  }
  //...
}
```

Introducere în Spring

//varianta 2

```
class ConcursService{
   private ParticipantiRepositoryFile repo;
  public ConcursService(){
     repo=new ParticipantiRepositoryFile("Participanti.txt");
  }
}
```

//varianta 2a

```
 public ConcursService(){
     repo=new ParticipantiRepositoryFile("Participanti2.txt",
                                          new ParticipantValidator());
  }
```

//varianta 3

```
class ConcursService{
   private ParticipantiRepositoryJdbc repo;
  public ConcursService(){
     Properties props=...
     repo=new ParticipantiRepositoryJdbc(props);
  }
}
```

Introducere în Spring

- Spring este un framework open-source, creat inițial de Rod Johnson și descris în cartea sa, *Expert One-on-One: J2EE Design and Development*.
- Frameworkul Spring a fost creat pentru a facilita dezvoltarea aplicațiilor complexe și foarte mari.
- În Spring se pot folosi obiecte simple Java (*POJO*), pentru a crea aplicații care anterior erau posibile doar folosind EJB.
- Un bean Spring este orice clasă Java (nu respectă regulile Java Beans).
- Spring promovează cuplarea slaba prin “injectarea” asocierilor și folosirea interfețelor.
- Spring folosește principiul IoC pentru “injectarea” asocierilor/ dependențelor.