Génie Logiciel
Gestion de données persistantes en C#

BAPTISTE PESQUET

Exemples de code

https://github.com/ensc-glog/persistance-csharp/tree/master/examples

Sommaire

- Introduction à la persistance
- Interactions avec les fichiers
- Accès à une base de données relationnelle
- Couche d'accès aux données (DAL)
- Mapping objet/relationnel (ORM)

Introduction à la persistance

La notion de donnée persistante

Donnée **persistante** : ne disparaît pas entre deux utilisations de l'application

Contraire : donnée **volatile**

Indispensable pour une très grande majorité des applications (données, configuration, etc)

Le stockage de données persistantes nécessite :

- Un support dédié, appelé parfois **mémoire de masse** (disque dur/SSD, mémoire Flash, etc)
- Un format adapté

Interactions avec les fichiers

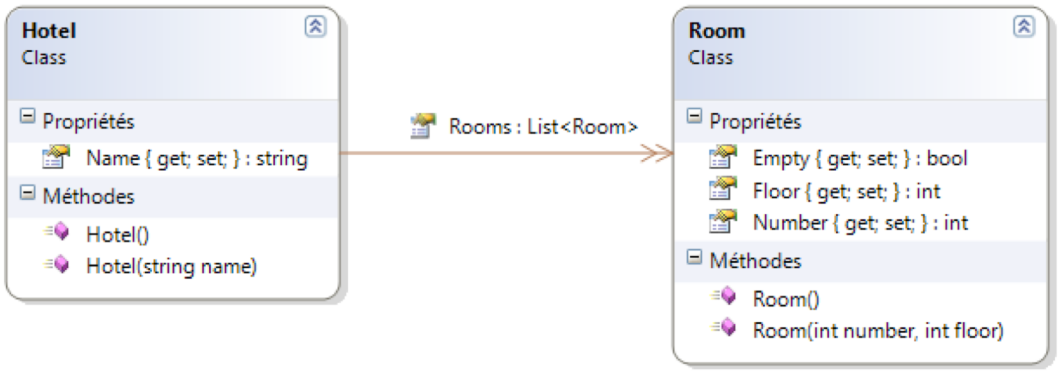
Notion de sérialisation

Sérialisation : transformation d'une information en mémoire sous une forme permettant son stockage persistant

Différents formats possibles :

- Binaire
- XML
- JSON

Contexte d'exemple



Le format XML

XML = **eXtensible Markup Language**

Format structuré par des balises ouvrantes et fermantes, un peu comme HTML

On peut définir ses propres balises => **métalangage**

Avantages :

- Lisibilité par un humain (!= binaire)
- Interopérabilité (échanges de données entre systèmes hétérogènes)

Inconvénient : verbosité

Exemple de document XML

```
<?xml version="1.0" encoding="utf-8"?>
<Hotel xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Name>Chelsea Hotel</Name>
  <Rooms>
    <Room> <Number>1</Number> <Floor>0</Floor> <Empty>true</Empty> </Room>
    <Room> <Number>2</Number> <Floor>1</Floor> <Empty>true</Empty> </Room>
    <Room> <Number>3</Number> <Floor>1</Floor> <Empty>false</Empty> </Room>
  </Rooms>
</Hotel>
```

Sérialisation vers un fichier XML

```
using System.IO;
using System.Xml;
using System.Xml.Serialization;
// …
// Sauvegarde l'objet hotel dans le fichier "hotel.xml"
StreamWriter writer = new StreamWriter("hotel.xml");
new XmlSerializer(typeof(Hotel)).Serialize(writer, hotel);
writer.Close();
```


Désérialisation depuis un fichier XML

```
using System.IO;
using System.Xml;
using System.Xml.Serialization;
// …
// Charge le contenu du fichier "hotel.xml" dans l'objet hotel
StreamReader reader = new StreamReader("hotel.xml");
Hotel hotel = (Hotel) new
XmlSerializer(typeof(Hotel)).Deserialize(reader);
reader.Close();
```

Le format JSON

JSON = **JavaScript Object Notation**

Format de description de données structurées inspiré de la syntaxe des objets JavaScript

Contenu JSON = ensemble de paires champ/valeur

Types de valeur possibles : nombres, chaînes, booléens, tableaux, objets

JSON a supplanté XML comme format standard pour les échanges de données sur le Web (utilisation d'API)

Exemple de document JSON

```
{
   "Name": "Chelsea Hotel",
   "Rooms": [
      { "Number": 1, "Floor": 0, "Empty": true },
      { "Number": 2, "Floor": 1, "Empty": true },
      { "Number": 3, "Floor": 1, "Empty": false }
   ]
}
```

JSON et .NET

Contrairement à XML, JSON n'est pas supporté nativement par le framework .NET

=> Nécessité d'intégrer une librairie externe

Standard : librairie **Newtonsoft.Json** accessible via le gestionnaire de packages **NuGet**

Sérialisation vers un fichier JSON

```
using System.IO;
using Newtonsoft.Json;
// …
// Sauvegarde l'objet hotel dans le fichier "hotel.json"
StreamWriter writer = new StreamWriter("hotel.json");
new JsonSerializer().Serialize(writer, hotel);
writer.Close();
```

Désérialisation depuis un fichier JSON

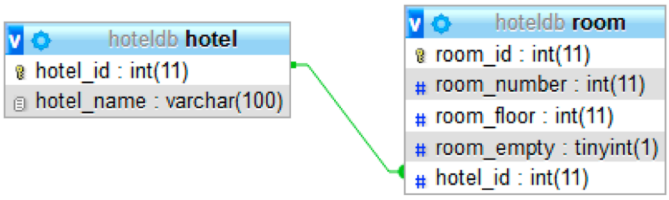
```
using System.IO;
using Newtonsoft.Json;
// …
// Charge le contenu du fichier "hotel.json" dans l'objet hotel
StreamReader reader = new StreamReader("hotel.json");
Hotel hotel = (Hotel) new JsonSerializer().Deserialize(reader, typeof(Hotel));
reader.Close();
```


Accès à une base de données relationnelle

Les avantages des SGBDR

- Technologie bien connue (invention dans les années 1970)
- Fiabilité
- Capacité à gérer de forts volumes de données
- Sécurité (authentification, réplication, etc)
- Langage d'interrogation (SQL) standard
- Offre logicielle riche : de ORACLE à SQLite en passant par PostgreSQL, MySQL/MariaDB ou SQL Server

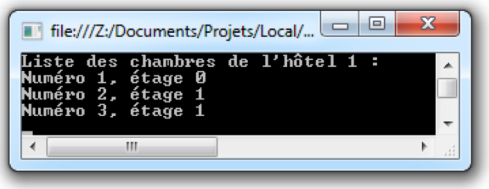
Contexte d'exemple



Connexion à MySQL

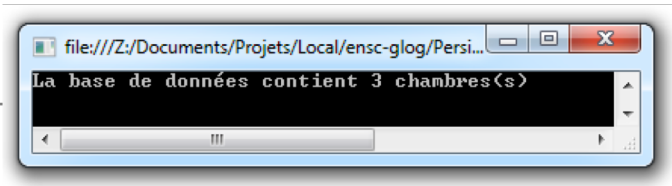
```
// Utilisation de la librairie MySQL.Data obtenue via NuGet
using MySql.Data.MySqlClient;
// …
string chaineCnx = 
"server=localhost;database=nomBD;user=nomUtil;password=mdpUtil;";
MySqlConnection connexion = new MySqlConnection(chaineCnx);
connexion.Open();
// Interactions avec la BD
// …
connexion.Close();
```


Lecture de données




```
// Connexion à MySQL
// …
MySqlCommand cmd = new MySqlCommand("SELECT * FROM room WHERE hotel_id=@IdHotel", 
connexion);
cmd.Parameters.AddWithValue("@IdHotel", 1);
MySqlDataReader reader = cmd.ExecuteReader();
if (reader.HasRows) {
    Console.WriteLine("Liste des chambres de l'hôtel 1 :");
    while (reader.Read())
        Console.WriteLine("Numéro " + reader["room_number"] + ", étage " + 
reader["room_floor"]);
}
reader.Close();
// Fermeture de la connexion
// …
```

Opérateurs SQL




```
// Connexion à MySQL
// …
MySQLCommand cmd = new MySqlCommand("SELECT COUNT(*) FROM room", connexion);
int nbChambres = Convert.ToInt32(cmd.ExecuteScalar());
Console.WriteLine("La base de données contient " + nbChambres + " chambres(s)");
// Fermeture de la connexion
// …
```


Ecriture de données

```
// Connexion à MySQL
// …
MySQLCommand cmd = new MySqlCommand("UPDATE room SET room_empty=@Empty WHERE room_number=@Number", connexion);
cmd.Parameters.AddWithValue("@Number", 2);
cmd.Parameters.AddWithValue("@Empty", false);
cmd.ExecuteNonQuery();
// Fermeture de la connexion
// …
```

Couche d'accès aux données (DAL)

La notion de DAL

DAL = ***Data Access Layer***

Partie d'un logiciel chargée de gérer l'accès aux données persistantes

Application du principe de séparation des responsabilités

Objectifs :

- Fournir une API uniforme pour accéder aux données
- Encapsuler la technologie d'accès

Expose des **interfaces** et masque les implémentations concrètes

=> modularité et évolutions facilitées

La notion de Repository

Abstraction vers une source de données pour une classe métier

Couple interface/implémentation(s)

Fournit des méthodes d'accès comme :

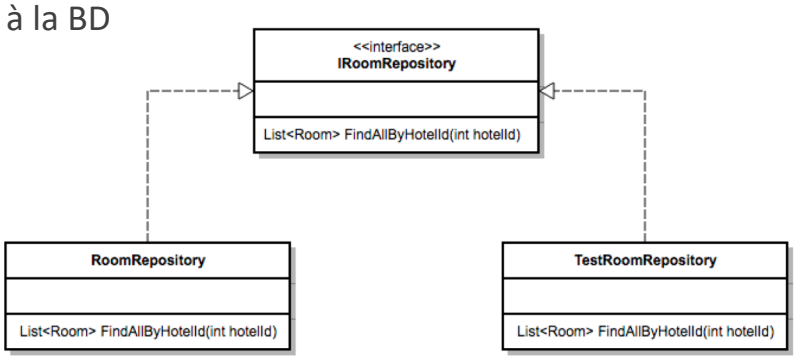
- GetAll()
- FindById()
- Add()
- Update()

- Proche de la notion de **DAO** (*Data Access Object*)

Exemple : repository pour la classe Room

RoomRepository : classe accédant à la base de données

TestRoomRepository : classe renvoyant des données de test sans accéder à la BD



Exemple : RoomRepository

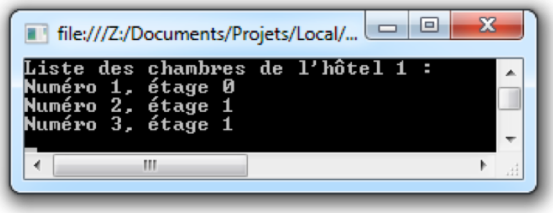
```
public List<Room> FindAllByHotelId(int hotelId)  {
   List<Room> rooms = new List<Room>();
   // … Connexion à MySQL et création de la requête
   while (reader.Read()) {
      Room room = new Room();
      room.Number = Convert.ToInt32(reader["room_number"]);
      room.Floor = Convert.ToInt32(reader["room_floor"]);
      room.Empty = Convert.ToBoolean(reader["room_empty"]);
      rooms.Add(room);
   }
   // … Fermeture du reader et de la connexion
   return rooms;
}
```


Exemple : TestRoomRepository

```
public List<Room> FindAllByHotelId(int hotelId)  {
   List<Room> rooms = new List<Room>();
   rooms.Add(new Room(212, 2));
   rooms.Add(new Room(213, 2));
   rooms.Add(new Room(312, 3));
   rooms[2].Empty = false;
   rooms.Add(new Room(412, 4));
   return rooms;
}
```

Exemple : utilisation d'un repository

```
IRoomRepository roomRepository = new RoomRepository();
// IRoomRepository roomRepository = new TestRoomRepository();
// …
List<Room> rooms = roomRepository.FindAllByHotelId(1);
string message = "Liste des chambres de l'hôtel 1 :\n";
foreach(Room room in rooms)
    message += "Numéro " + room.Number + ", étage " + room.Floor + "\n";
Console.WriteLine(message);
```



Mapping objet/relationnel (ORM)

ORM ?

Object/Relational Mapping

SGBDR : standard actuel pour la persistance des données

POO : standard actuel pour la création d'applications

=> ORM (parfois écrit O/RM) : lien entre la base de données et les objets manipulés par l'application

Le modèle relationnel

Unité de représentation : table

Caractéristiques : colonnes

Entité : tuple (ligne)

Types de données : simples (nombres, chaîne, date, etc)

Identifiant : explicite (clé primaire)

Relations entre éléments : jointures SQL

Concepts spécifiques :

- Déclencheurs
- Procédures stockées
- …

Le modèle objet

Unité de représentation : classe

Caractéristiques : propriétés et méthodes

Entité : objet

Types de données : simples ou complexes

Identifiant : implicite (référence mémoire)

Relations entre éléments : navigation des associations

Concepts spécifiques :
- Encapsulation
- Héritage
- Interfaces
- …

Un problème complexe

Deux modèles très éloignés (***impedance mismatch***)

Pas de solution idéale : « ORM is the Vietnam of computer science »

Solution 1 : supprimer le SGBDR

Stocker les données sous forme d'objets (SGBDOO)

Utiliser des services intermédiaires au lieu d'attaquer directement une BD (API, cloud, etc)

Solution 2 : supprimer les objets

Revenir à la programmation procédurale

Manipuler des types de bases et des tableaux/listes

Adopter un autre paradigme comme la programmation **fonctionnelle** (Haskell, Clojure, Kotlin, OCaml, F#, JavaScript, etc)

Solution 3 : ORM manuel

Lien manuel entre la BD et les objets :

- Requêtage SQL
- Parcours des résultats
- Création d'objets à partir des résultats
- Renvoi de listes d'objets

Exemple : HotelRepository

```
public List<Hotel> GetAll() {
    List<Hotel> hotels = new List<Hotel>();
    // … Connexion à MySQL et création du DataReader
    while (reader.Read()) {
        Hotel hotel = new Hotel(reader["hotel_name"].ToString());
        hotel.Id = Convert.ToInt32(reader["hotel_id"]);
        hotel.Rooms = roomRepository.FindAllByHotelId(hotel.Id);
        hotels.Add(hotel);
    }
    // … Fermeture du DataReader et de la connexion
    return hotels;
}
```

Avantages/Inconvénients

Avantages

- Simplicité
- Solution sur mesure : pas de mauvaises surprises liées à des « boîtes noires »
- Exploitation possible de toutes les possibilités de SQL

Inconvénients

- Volume de code (*boilerplate*)
- Répétition des mêmes problématiques (transformation résultats SQL/objets, etc)
- Aspects complexes : gestion du pool de connexions, du cache, etc

Solution 3 : ORM automatisé

Intégrer un outil dédié à l'ORM

Offre logicielle riche et multiplateforme :

- C# : Entity Framework, NHibernate
- Java : Persistence API, Hibernate
- PHP : Doctrine (Symfony), Eloquent (Laravel)
- Python : Django
- …

Principe de fonctionnement

Approche générale :

- On fournit à l'outil d'ORM les correspondances entre classes et tables (***mapping***) sous la forme de fichiers de configuration ou d'annotations
- Durant l'exécution, l'outil génère automatiquement les requêtes SQL nécessaires

Certains outils d'ORM peuvent faire plus :

- On crée soit les classes métier (« code first »), soit le schéma relationnel (« DB first »)
- L'outil génère automatiquement le schéma BD ou les classes métier
- Les modifications sont synchronisées entre les deux

Avantages/Inconvénients

Avantages

- Automatisation de la gestion du mapping
- Diminution du volume de code
- Mise en œuvre de bonnes pratiques
- Performances (le plus souvent)

Inconvénients

- Nécessite de maitriser un nouvel outil
- Certaines limitations liées à l'outil
- Aspect « boîte noire »
- Performances dans certains cas