Modern Architecture

Speaker: Luca Minuti - Maurizio Del Magno

DelphiDay italian conference

Modern Architecture

Luca Minuti
Freelance

WiRL github.com/delphi-blocks/WiRL

OpenSSL github.com/lminuti/Delphi-OpenSSL

luca.minuti@gmail.com

facebook.com/lithian

Maurizio Del Magno
Developer

Levante Software

i-ORM github.com/mauriziodm/iORM
DJSON github.com/mauriziodm/DJSON

mauriziodm@levantesw.it
mauriziodelmagno@gmail.com

facebook.com/maurizio.delmagno
iORM + DJSON (group)

DelphiDay italian conference

Modern Architecture

Problema

I requisiti cambiano...

Il mio codice è facilmente manutenibile/estendibile/riutilizzabile?

oppure…

- **Rigidità**
  - resistenza cambiamento
  - ogni modifica causa una cascata di ulteriori modifiche
- **Fragilità**
  - tendenza a "rompersi" in diversi punti ad ogni modifica (anche senza una relazione concettuale con l'area che è cambiata)
- **Immobilità**
  - impossibilità di riutilizzo di codice da altri progetti o anche all'interno dello stesso

DI CHI È LA COLPA?

- La colpa è dei requisiti che cambiano!!!
- E se poniamo come primo requisito/specifica il fatto che <u>i requisiti cambieranno</u>?
- Allora è il nostro DESIGN ad essere SBAGLIATO...
- ...perché non è in grado di adattarsi ai costanti cambiamenti di specifiche/requisiti che già sappiamo avverranno

DI CHI È LA COLPA?

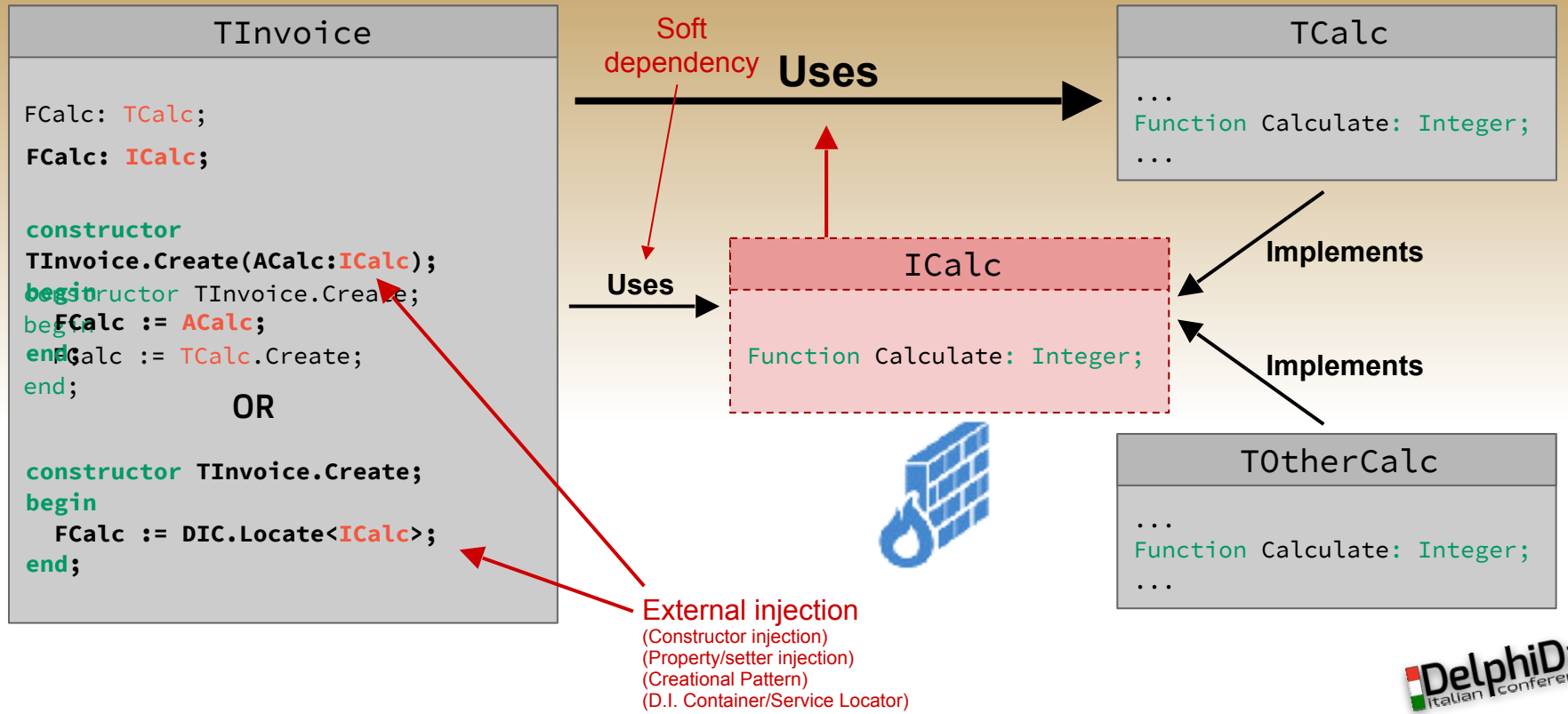
- La colpa è delle DIPENDENZE
- Dipendenze improprie tra i diversi moduli/strati della nostra applicazione sono la principale causa dei nostri problemi
- Il degrado dell'architettura delle dipendenze può causare l'impossibilità di mantenere il nostro software

Gestione delle dipendenze

- Le dipendenze vanno gestite
- Creazione di “firewalls” per dipendenze
- I firewalls impediscono la diffusione delle dipendenze tra i vari moduli/strati dell’applicazione
- Firewalls = interfacce

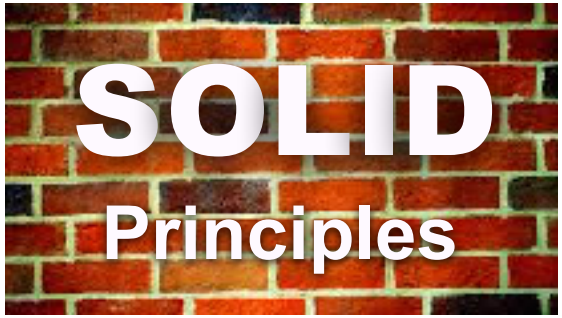
Cos'è una Dipendenza?

Dependency Injection



DelphiDay Italian Conference

Le interfacce sono nostre amiche



Bastano?

Serve anche una buona architettura/design

Esistono dei principi e/o delle tecniche a cui ispirarsi?

DelphiDay Italian conference

Object Oriented Design

Single Responsibility Principle *(SRP)*

Open Closed Principle *(OCP)*

Liskov Substitution Principle *(LSP)*

Interface Segregation Principle *(ISP)*

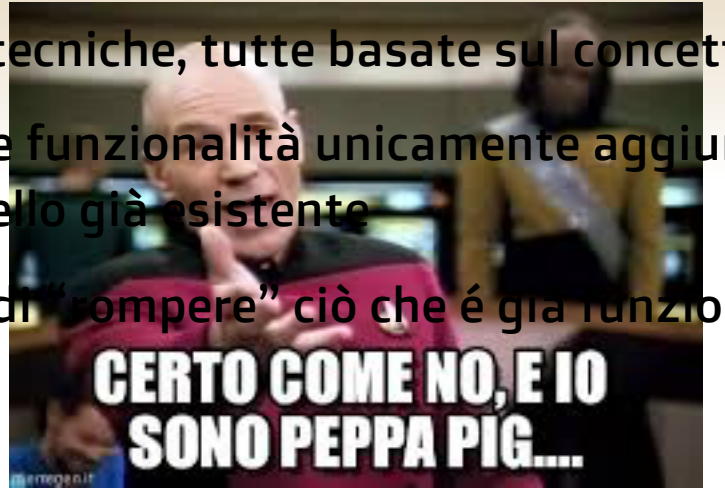
Dependency Inversion Principle *(DIP)*

Single Responsibility Principle *(SRP)*

- Non ci dovrebbe mai essere più di una ragione per dover modificare una classe
- Ogni responsabilità è un possibile motivo di cambiamento
- Se una classe assume più di una responsabilità allora queste responsabilità sono “accoppiate” (coupled)
- Questo tipo di accoppiamento porta a modelli fragili che si rompono in modi inaspettati quando vengono cambiati

Open Closed Principle *(OCP)*

- E' il più importante
- Un modulo deve essere aperto alle estensioni ma chiuso alle modifiche
- Come dire che vogliamo essere in grado di cambiare quello che un modulo fa, senza cambiare il codice sorgente del modulo stesso
- Esistono diverse tecniche, tutte basate sul concetto di "astrazione"
- Aggiungere nuove funzionalità unicamente aggiungendo codice nuovo, senza toccare quello già esistente
- ... così eviteremo di "rompere" ciò che é già funzionante e testato in precedenza



Liskov Substitution Principle *(LSP)*

- Ogni classe deve poter essere sostituita da qualunque sua derivata senza che il codice utilizzatore cambi il suo comportamento o cessi di funzionare
- Se una classe non è conforme il codice utilizzatore è costretto a “conoscere” (e quindi discernere) anche tutte le sue derivate
- Questo causa anche la violazione del “Open Close Principle”
  - il codice utilizzatore deve essere modificato ogni volta che una nuova classe derivata viene creata

Code sample

```
function DrawShape(Shape: TShape)
begin
  if Shape is TSquare then
    DrawSquare(TSquare(Shape))
  else
  if Shape is TCircle then
    DrawCircle(TCircle(Shape));
end;
```

… e quando dovremo aggiungere un altro poligono?

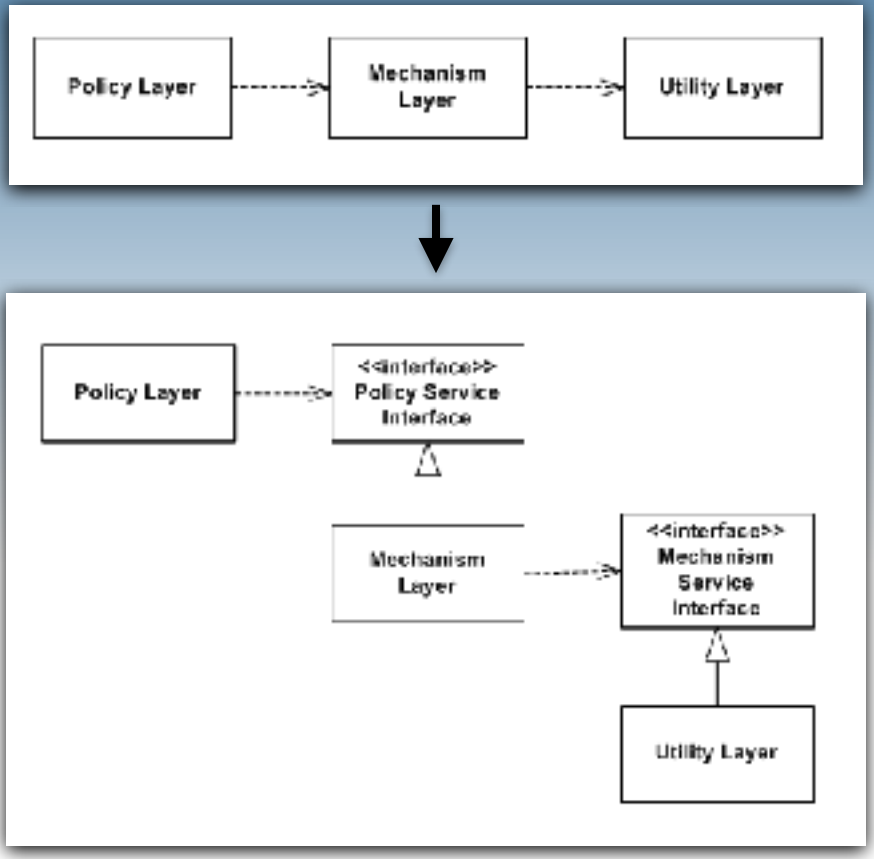
Interface Segregation Principle *(ISP)*

- Più interfacce piccole e specifiche sono meglio di una sola grande generale interfaccia
- Ciascun client non dovrebbe dipendere da metodi e funzionalità che non usa effettivamente
- Un oggetto dovrebbe implementare numerose interfacce, una per ciascun ruolo che l'oggetto stesso gioca in diversi contesti e relazioni con altri oggetti

Dependency Inversion Principle *(DIP)*

- Dipendi dalla astrazioni e non dalle implementazioni
- Di solito i moduli di alto livello* realizzano le proprie funzioni facendo uso di componenti di più basso livello
- I moduli di alto livello non devo dipendere da quelli di basso livello, entrambi devono dipendere solo da astrazioni
- Il riferimento alle astrazioni però è diverso nei due casi
- I moduli di altro livello “usano” tali astrazioni
- Quelli di basso livello le “implementano"

Dependency Inversion Principle *(DIP)*



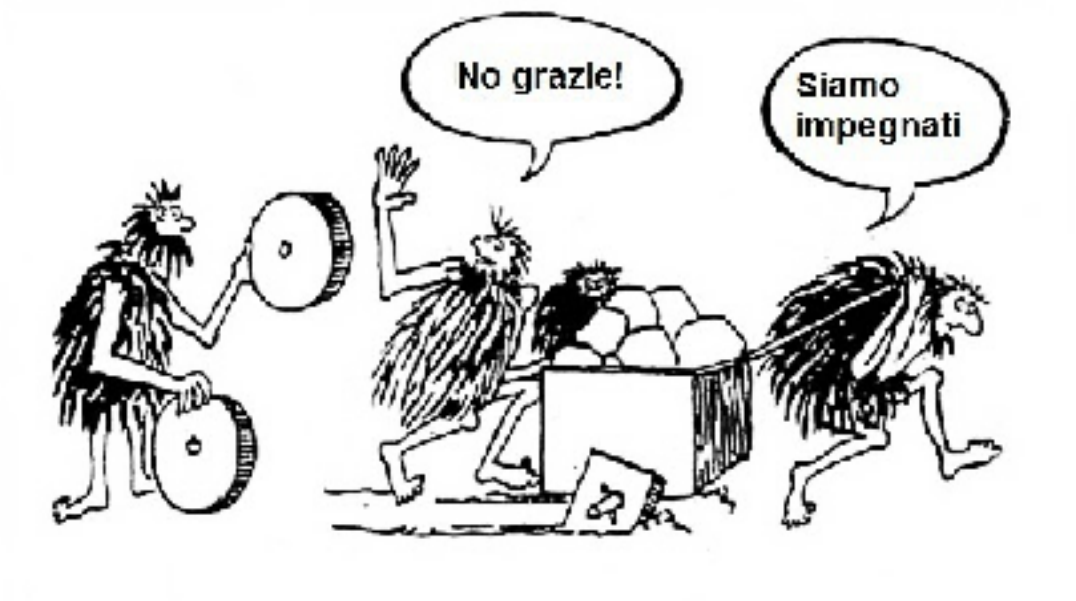
Object Oriented Design

- Progettare software O.O. non è facile, farlo in modo che sia riusabile è ancora più difficile
  - Responsabilità, granularità, ereditarietà ecc.
- Il sistema dovrebbe:
  - essere specifico per il problema di oggi
  - ma sufficientemente generale per adattarsi anche ad esigenze future
  - avere un'architettura tale da minimizzare, o almeno centralizzare, eventuali redesign
- E' difficile progettare architetture abbastanza riusabili e flessibili al primo tentativo
- Prima di raggiungere un buon risultato si reitera spesso sulle scelte progettuali
- All'inizio si fa confusione fra tutte le opzioni disponibili
  - "utilizzo un'interfaccia oppure una classe astratta?"
  - "riutilizzo tramite ereditarietà oppure per composizione?"
  - "come assegno e/o divido le responsabilità?"

Object Oriented Design

- Non cercare di reinventare la ruota
- Molte persone hanno già affrontato lo stesso problema
- Usa soluzioni che si sono già dimostrate efficaci in precedenza
- I “Design Patterns” sono una libreria di soluzioni architetturali che hanno già dimostrato di essere valide

Design Pattern?



Cosa Sono?

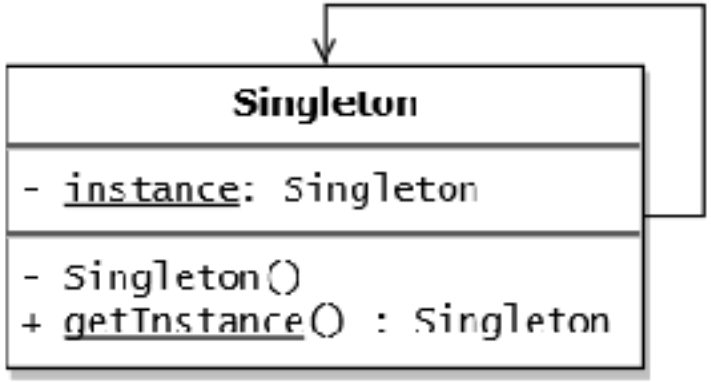
- Una soluzione generale a un problema ricorrente
- Non sono una libreria di codice o un componente riutilizzabile
- Uno schema architetturale per risolvere un problema
- Categorie:
  - Pattern creazionali (abstract factory, builder, singleton...)
  - Pattern strutturali (adapter, proxy, facade...)
  - Pattern comportamentali (command, observer, iterator...)
  - Pattern architetturali (client-server, MVC, MVVM...)
  - Altri...

DelphiDay italian conference

Singleton

- Scopo:
  - Assicurare che per una determinata classe esista una unica istanza attiva, fornendo un unico punto di accesso globale per accedervi
- Quando:
  - utile quando si ha la necessità di centralizzare informazioni e/o comportamenti in un'unica entità condivisa da tutti i suoi utilizzatori
- Come:
  - la soluzione più adatta a risolvere le esigenze di questo pattern (unicità dell'istanza) consiste nell'associare alla classe stessa la responsabilità di creare la sua istanza
  - in questo modo è la classe stessa che assicura che nessun'altra istanza possa essere creata, intercettando e gestendo in modo centralizzato le richieste
  - di solito è presente un metodo "getter" statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata (lazy initialization)

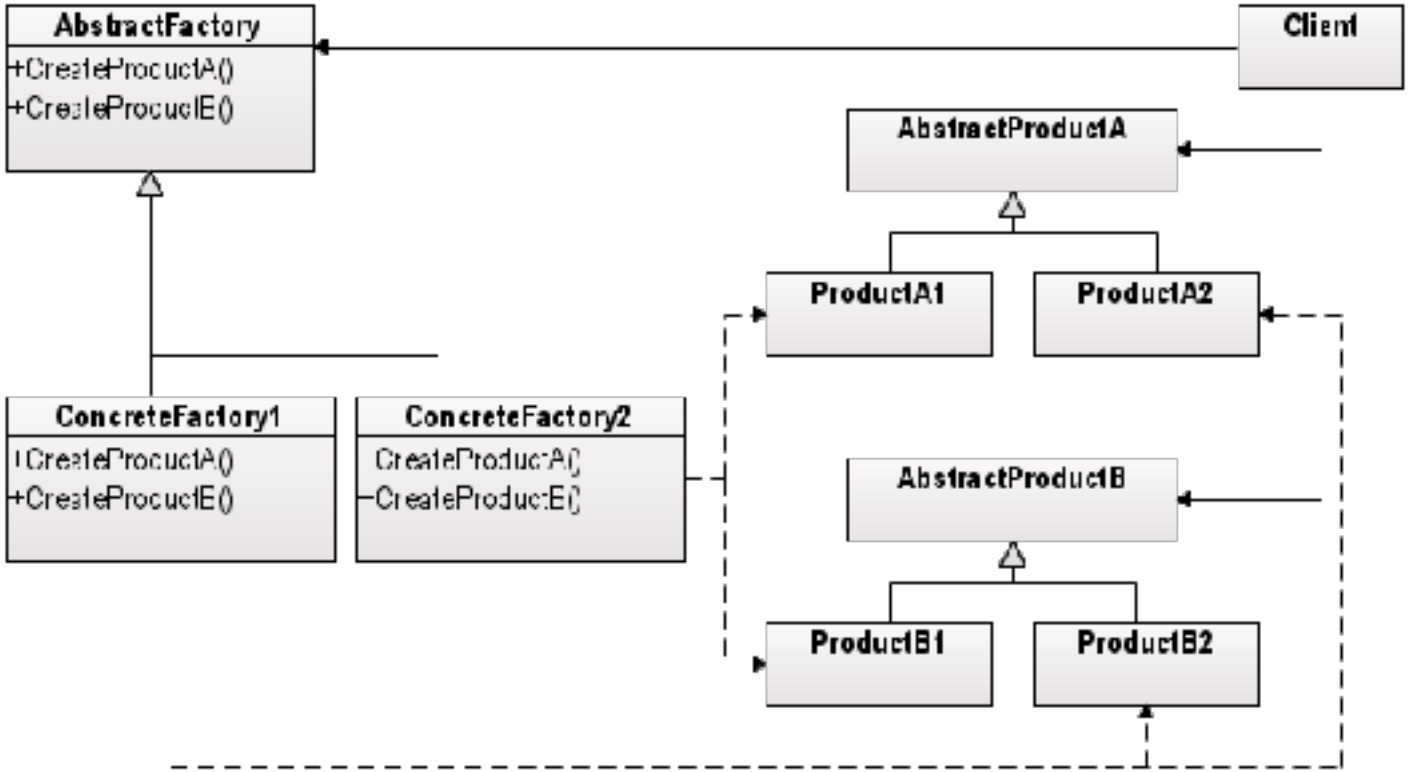
Singleton



Abstract Factory

- Scopo:
  - Fornire un'interfaccia astratta per la creazione di famiglie di oggetti tra loro correlati (o dipendenti) limitando l'accoppiamento derivante dall'uso diretto delle classi concrete
- Quando si vuole...
  - un sistema indipendente da come gli oggetti vengono creati, composti e rappresentati
  - permettere la configurazione del sistema come scelta fra diverse famiglie di oggetti
  - che gli oggetti, che sono organizzati in famiglie, siano vincolati ad essere utilizzati con altri della stessa famiglia
  - fornire una libreria di classe mostrando solo le interfacce e nascondendo le implementazioni
- Come:
  - di solito si crea una sola istanza "concreta" a run-time
  - una istanza gestisce la creazione di una sola famiglia di oggetti con una implementazione specifica
  - per creare oggetti di un'altra famiglia bisogna istanziare un'altra factory "concreta"
- Vantaggi:
  - consente di cambiare in modo semplice la famiglia di oggetti utilizzata (anche a run-time)
  - promuove la coerenza perché consente di creare solo oggetti di una stessa famiglia

Abstract Factory



Dubbi ???

Demo time...

DelphiDay Italian conference

Code sample

```
procedure TForm1.Button1Click(Sender: Tobject);
var
   Tot: Currency;
begin
   DS.Open;
   Tot := 0;
   While not DS.eof do
   begin
      Tot := Tot + DS.FieldByName['Qta'].AsInteger                    zo'].AsCurrency;
      DS.Next;
   end;
   DS.Close;
   ShowMessage('Totale: ' + CurrToStr(Tot));
end;
```

Quindi se uso un DB non posso usare la OOP?

Questo è OOP ?

dove è il comportamento?
dove sono i dati?
dove è incapsulata la business logic?

Programma applicativo

Insieme organizzato di 3 strati funzionali

- **Presentation layer** (user interface)
- **Domain logic layer** (business logic)
- **Persistence layer** (data access logic)
-

Presentation Layer

User Interface

- Si occupa dell'interazione con l'utente
- Riceve i dati che l'utente fornisce come input
- Presenta, come output, i risultati dell'esecuzione del programma
- Cattura le intenzioni dell'utente e le esaudisce eseguendo uno o più comandi

Domain Logic Layer

Business Logic

- Sottosistema di logica applicativa
- Definisce, in forma di classi, le informazioni e gli specifici algoritmi di manipolazione e validazione che caratterizzano l'applicazione
- Es. per gestione ordini: TCliente, TOrdine, TRigaOrdine, TArticolo ecc.

Persistence Layer

Data Access Logic

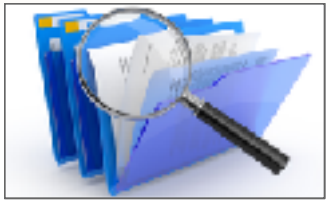
- Sottosistema di gestione della persistenza
- Si occupa dell'organizzazione dei dati, della loro memorizzazione e del loro successivo reperimento

Programma applicativo

Insieme organizzato di 3 strati funzionali

Presentation

Domain



Data access

Note:

Come **disponiamo** le logiche che compongono la nostra applicazione e come provvediamo alla loro reciproca **separazione** è di rilevante importanza.

Influenza la **manutenibilità**, **testabilità** e in generale la **qualità** del nostro lavoro.

In questo differiscono tra loro i pattern architetturali più conosciuti (MVC, MVP, PM, MVVM).

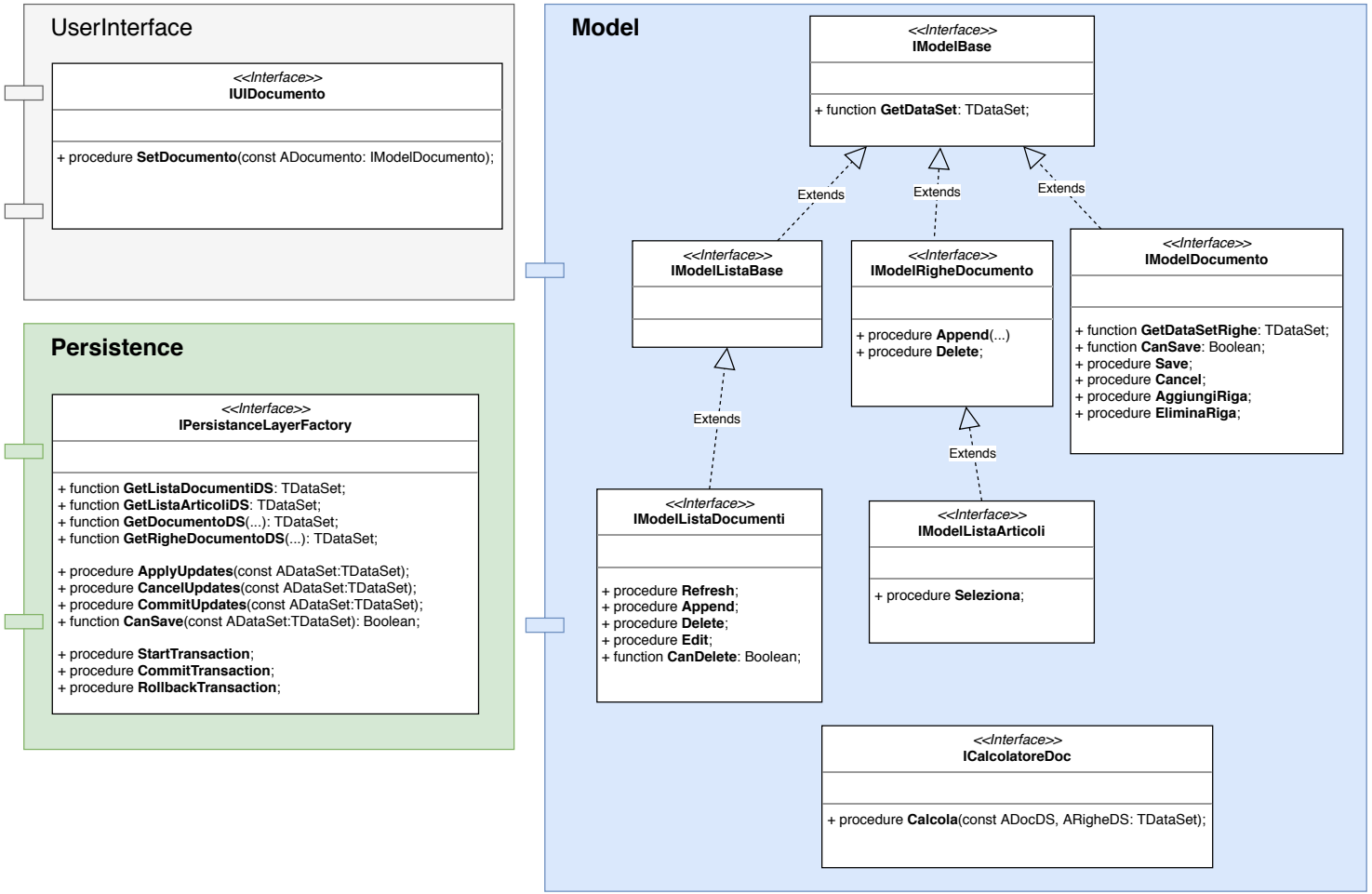
DelphiDay Italian conference

Dubbi ???

Demo time...

DelphiDay Italian conference

MicroGest


UserInterface
<<Interface>>
IUIDocumento
+ procedure SetDocumento(const ADocumento: IModelDocumento);
Persistence
<<Interface>>
IPersistanceLayerFactory
+ function GetListaDocumentiDS: TDataSet;
+ function GetListaArticoliDS: TDataSet;
+ function GetDocumentoDS(...): TDataSet;
+ function GetRigheDocumentoDS(...): TDataSet;
+ procedure ApplyUpdates(const ADataSet:TDataSet);
+ procedure CancelUpdates(const ADataSet:TDataSet);
+ procedure CommitUpdates(const ADataSet:TDataSet);
+ function CanSave(const ADataSet:TDataSet): Boolean;
+ procedure StartTransaction;
+ procedure CommitTransaction;
+ procedure RollbackTransaction;
Model
<<Interface>>
IModelBase
+ function GetDataSet: TDataSet;
Extends
Extends
Extends
<<Interface>>
IModelListaBase
<<Interface>>
IModelRigheDocumento
+ procedure Append(...)
+ procedure Delete;
<<Interface>>
IModelDocumento
+ function GetDataSetRighe: TDataSet;
+ function CanSave: Boolean;
+ procedure Save;
+ procedure Cancel;
+ procedure AggiungiRiga;
+ procedure EliminaRiga;
Extends
Extends
<<Interface>>
IModelListaDocumenti
+ procedure Refresh;
+ procedure Append;
+ procedure Delete;
+ procedure Edit;
+ function CanDelete: Boolean;
<<Interface>>
IModelListaArticoli
+ procedure Seleziona;
<<Interface>>
ICalcolatoreDoc
+ procedure Calcola(const ADocDS, ARigheDS: TDataSet);

Runtime Packages

- Un package non è nient'altro che un particolare tipo di libreria (dll)
- Sono usati dall'IDE ma anche dalla applicazioni
- Possono essere caricati staticamente o dinamicamente
- Possono contenere più unit (ma non hanno un entry point)
- Possono far riferimento ad altri package tramite la keyword require
- Attenzione: non possono contenere riferimenti circolari
- Attenzione: una unit non può far parte di due package (usati contemporaneamente)

Dubbi ???

Demo time...

DelphiDay Italian conference

Ancora dubbi ???

Fai una domanda...

DelphiDay Italian conference

Grazie !

Maurizio del Magno
Developer

Luca Minuti
Freelance

i-ORM github.com/mauriziodm/iORM
DJSON github.com/mauriziodm/DJSON

mauriziodm@levantesw.it
mauriziodelmagno@gmail.com

facebook.com/maurizio.delmagno
iORM + DJSON (group)

WiRL github.com/delphi-blocks/WiRL
OpenSSL github.com/lminuti/Delphi-OpenSSL

luca.minuti@gmail.com

facebook.com/lithian

DelphiDay italian conference

wintech italia