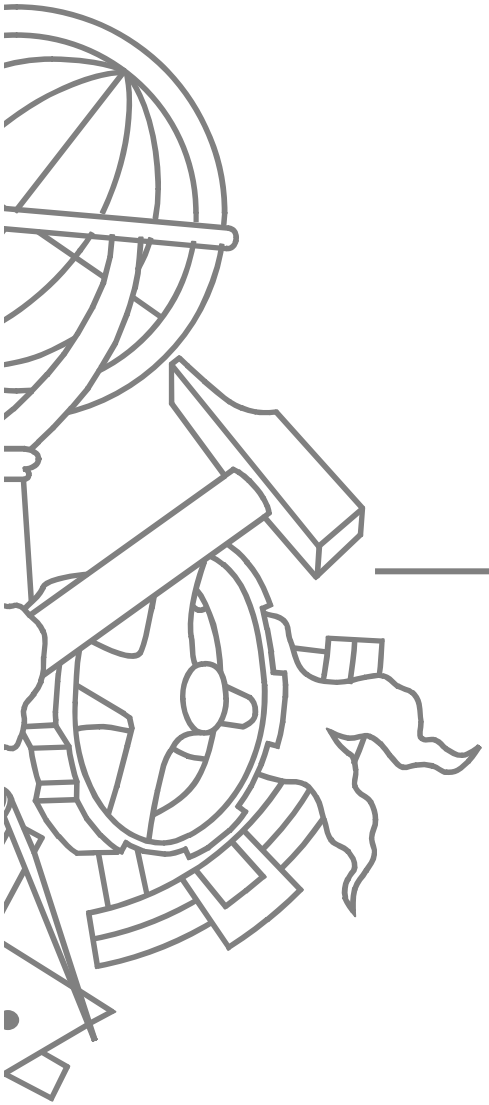
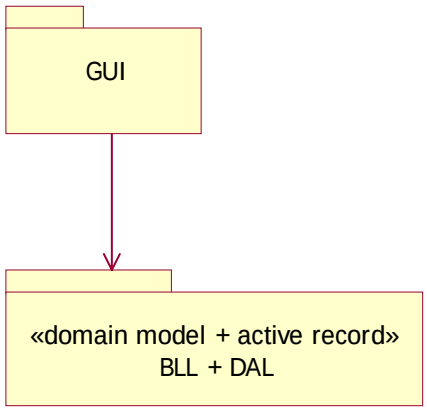
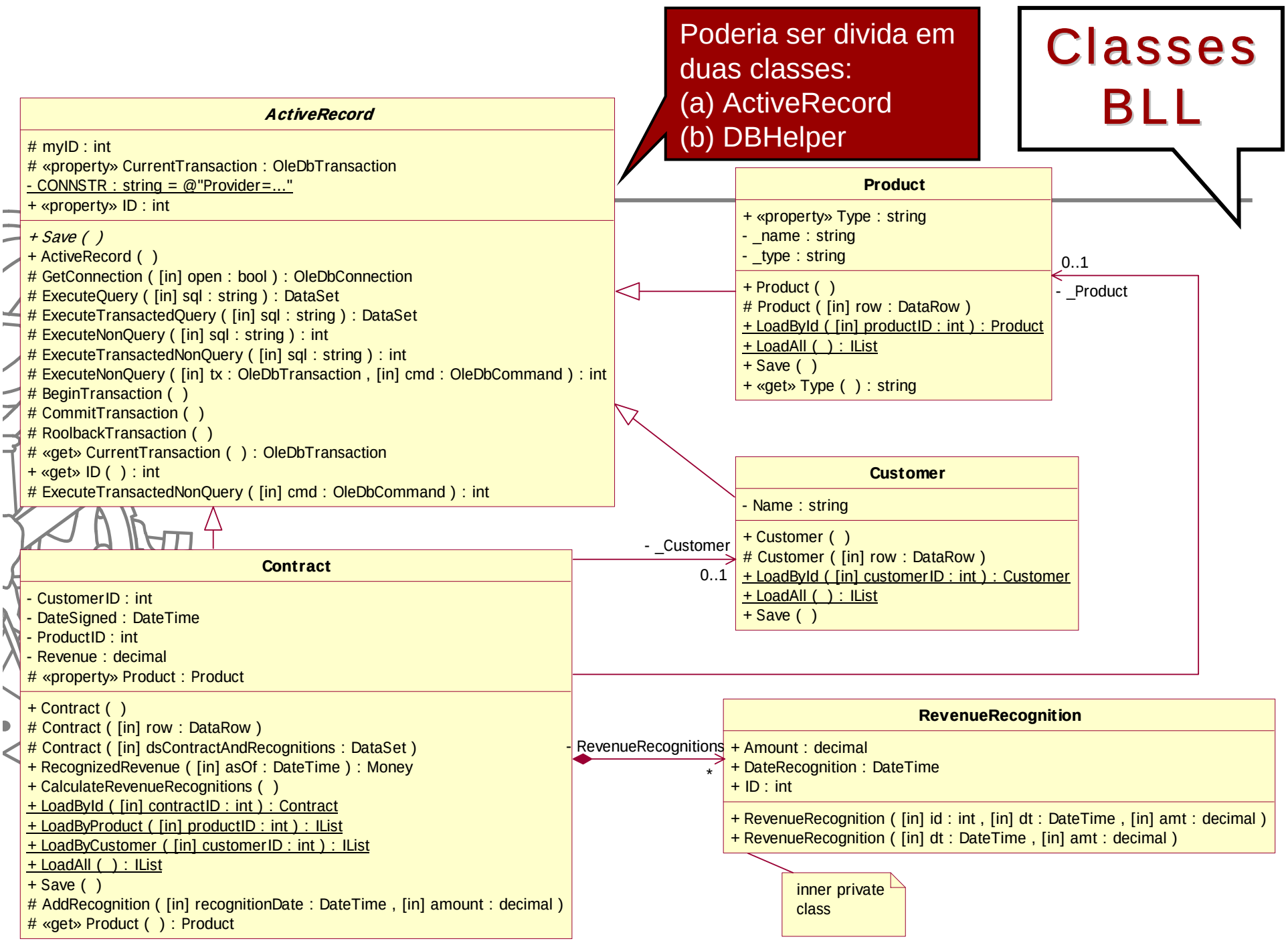


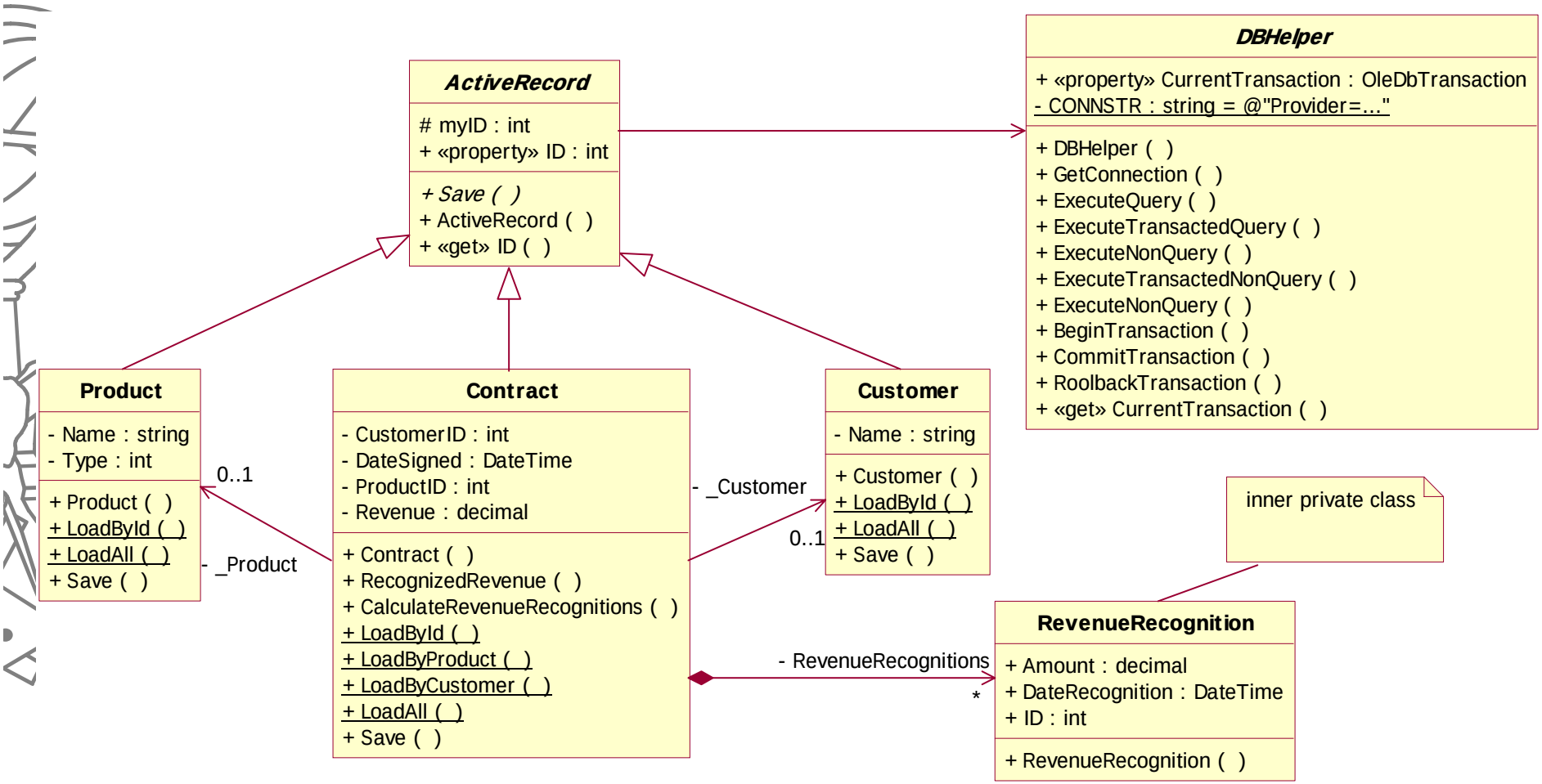
Domain Model com Active Record

Packages

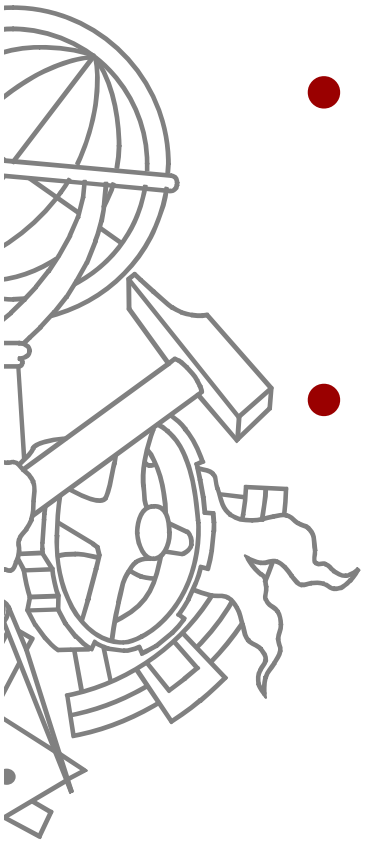



Poderia ser divida em duas classes:
(a) ActiveRecord
(b) DBHelper
Classes BLL
ActiveRecord
myID : int
«property» CurrentTransaction : OleDbTransaction
- CONNSTR : string = @"Provider=..."
+ «property» ID : int
+ Save ( )
+ ActiveRecord ( )
# GetConnection ([in] open : bool) : OleDbConnection
# ExecuteQuery ([in] sql : string) : DataSet
# ExecuteTransactedQuery ([in] sql : string) : DataSet
# ExecuteNonQuery ([in] sql : string) : int
# ExecuteTransactedNonQuery ([in] sql : string) : int
# ExecuteNonQuery ([in] tx : OleDbTransaction , [in] cmd : OleDbCommand) : int
# BeginTransaction ( )
# CommitTransaction ( )
# RoolbackTransaction ( )
# «get» CurrentTransaction ( ) : OleDbTransaction
+ «get» ID ( ) : int
# ExecuteTransactedNonQuery ([in] cmd : OleDbCommand) : int
Product
+ «property» Type : string
- _name : string
- _type : string
+ Product ( )
# Product ([in] row : DataRow)
+ LoadById ([in] productID : int) : Product
+ LoadAll ( ) : IList
+ Save ( )
+ «get» Type ( ) : string
0..1
- _Product
Customer
- Name : string
+ Customer ( )
# Customer ([in] row : DataRow)
+ LoadById ([in] customerID : int) : Customer
+ LoadAll ( ) : IList
+ Save ( )
- _Customer
0..1
Contract
- CustomerID : int
- DateSigned : DateTime
- ProductID : int
- Revenue : decimal
«property» Product : Product
+ Contract ( )
# Contract ([in] row : DataRow)
# Contract ([in] dsContractAndRecognitions : DataSet)
+ RecognizedRevenue ([in] asOf : DateTime) : Money
+ CalculateRevenueRecognitions ( )
+ LoadById ([in] contractID : int) : Contract
+ LoadByProduct ([in] productID : int) : IList
+ LoadByCustomer ([in] customerID : int) : IList
+ LoadAll ( ) : IList
+ Save ( )
# AddRecognition ([in] recognitionDate : DateTime , [in] amount : decimal)
# «get» Product ( ) : Product
- RevenueRecognitions
*
RevenueRecognition
+ Amount : decimal
+ DateRecognition : DateTime
+ ID : int
+ RevenueRecognition ([in] id : int , [in] dt : DateTime , [in] amt : decimal)
+ RevenueRecognition ([in] dt : DateTime , [in] amt : decimal)
inner private class

Classes (alternativa)

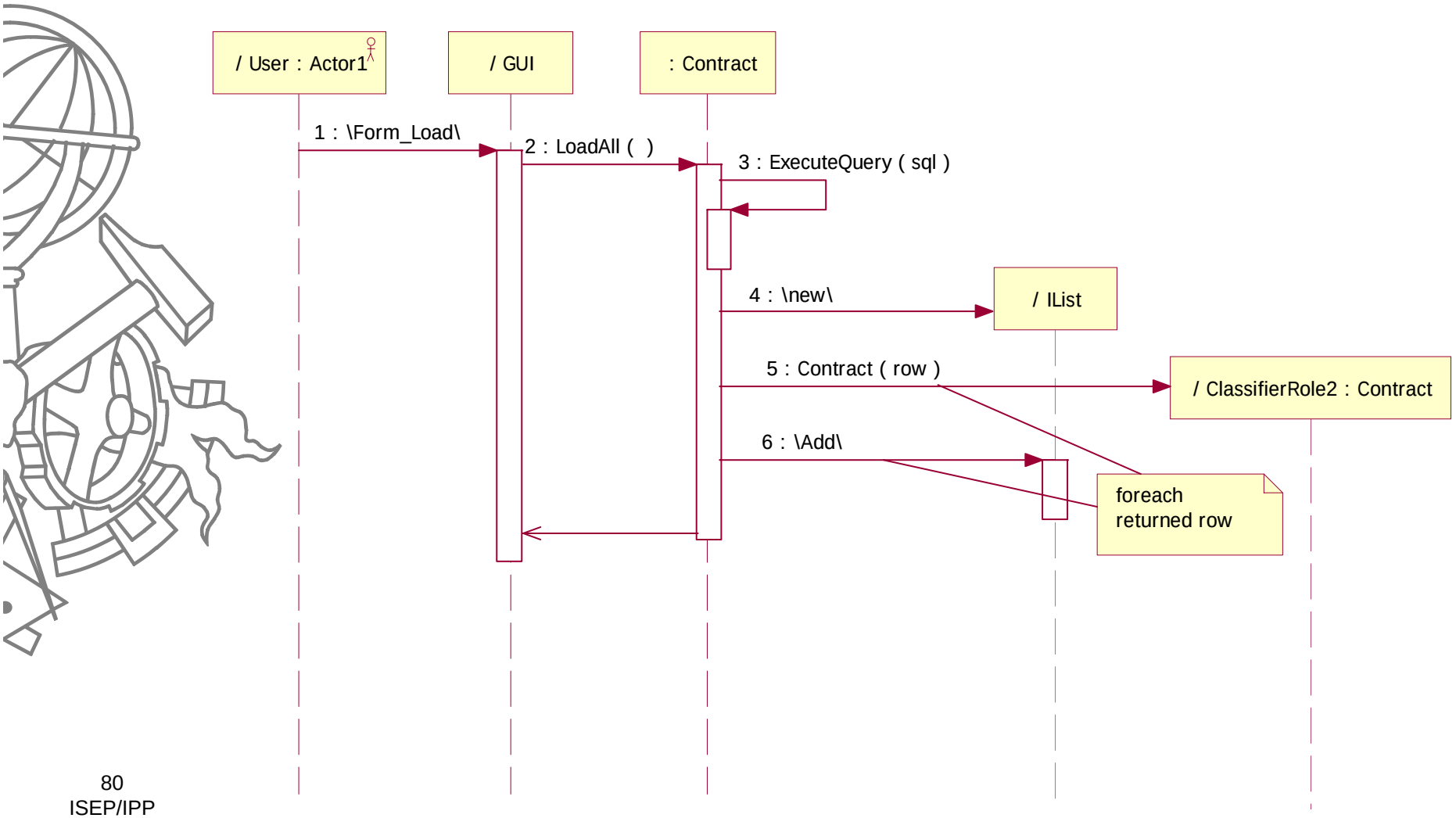


Algumas notas

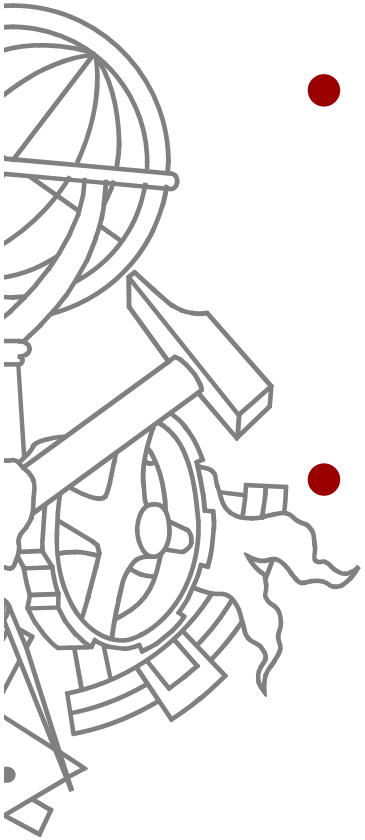


- Métodos *Finder* poderiam estar em classe separada em vez de serem métodos **`static`** na classe
- Se usar Lazy Load necessita obrigatoriamente de ter atributos com *ids* das chaves estrangeiras (ex, **`customerId`** na classe **`Contract`**)

Sequência GetContracts

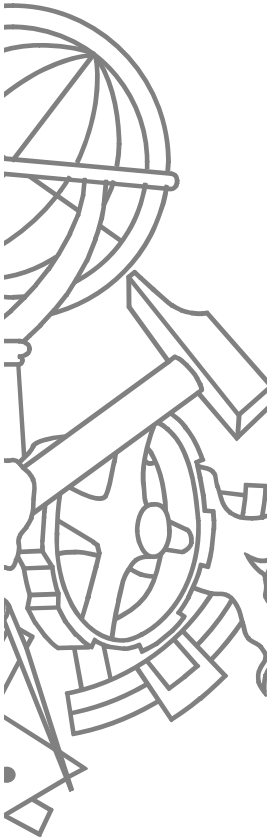


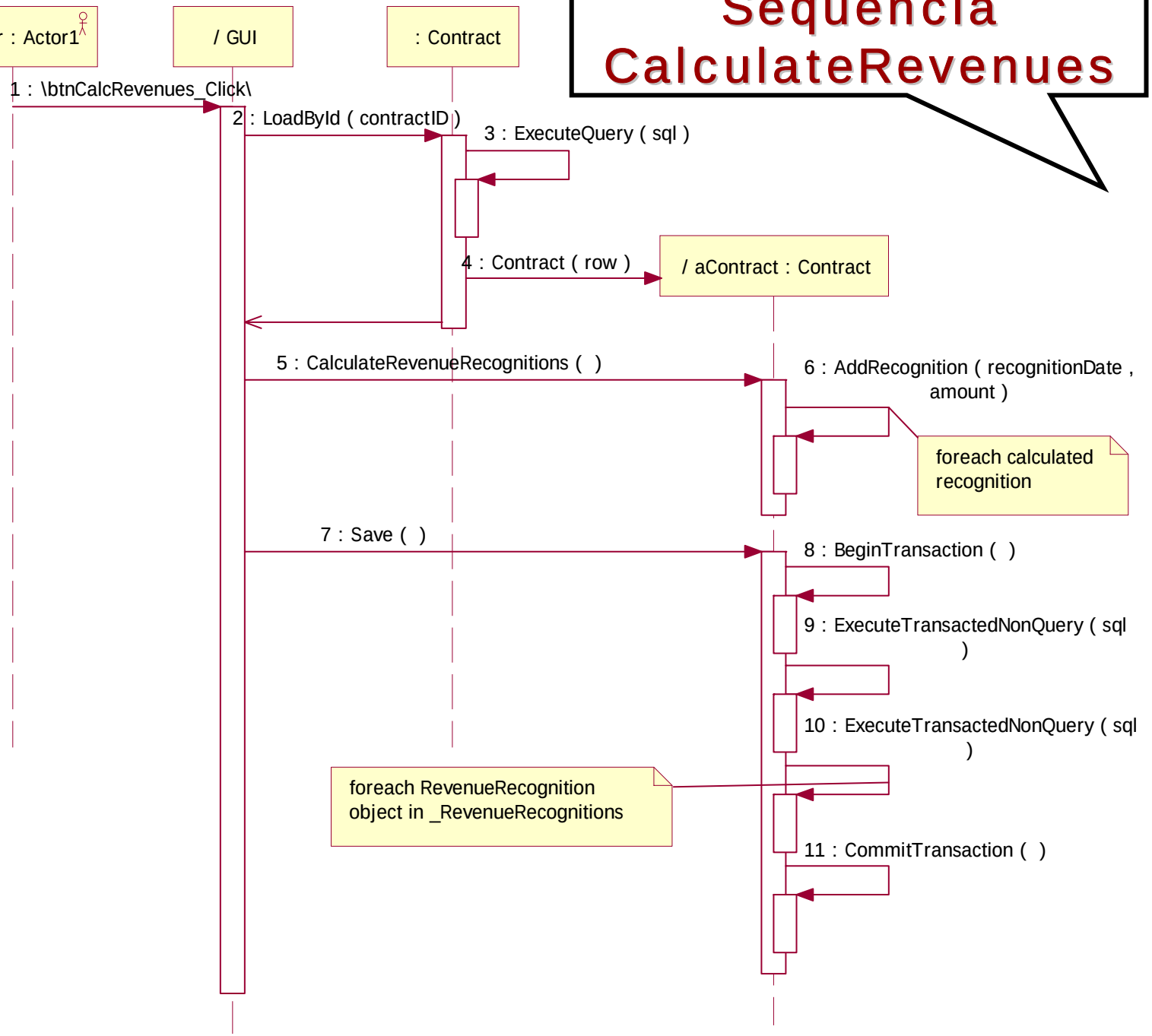
Domain Model com Active Record



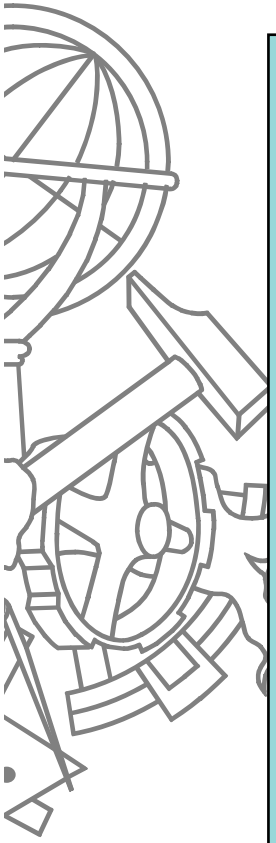
- Como fazer relativamente aos objectos **`RevenueRecognition`**?
  - Usar Lazy Load ou carregar logo para memória?
- E relativamente aos objectos **`Product`** referenciados nos contratos?
  - usar Lazy Load?
  - Deveria usar Identity Map

Sequência CalculateRevenues





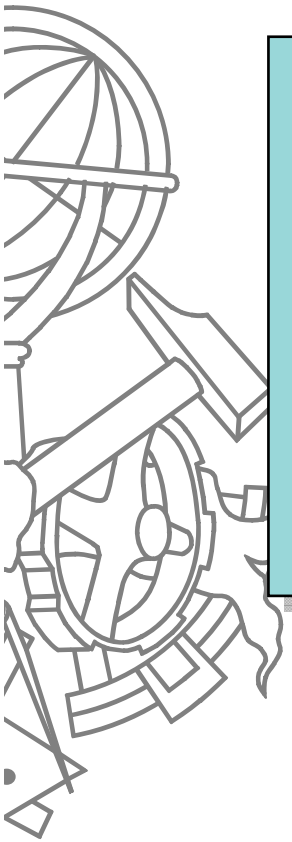
CalculateRevenues()



```
public void CalculateRevenueRecognitions()
{
    switch (this.Product.Type) {
        case "PT":
          AddRecognition(DateSigned, Revenue);
          break;
        case "FC":
          decimal[] allocsFC = Money.Allocate(Revenue, 3);
          AddRecognition(DateSigned, allocsFC[0]);
          AddRecognition(DateSigned.AddDays(60), allocsFC[1]);
          AddRecognition(DateSigned.AddDays(90), allocsFC[2]);
          break;
        case "BD":
          decimal[] allocsBD = Money.Allocate(Revenue, 3);
          AddRecognition(DateSigned, allocsBD[0]);
          AddRecognition(DateSigned.AddDays(30), allocsBD[1]);
          AddRecognition(DateSigned.AddDays(60), allocsBD[2]);
          break;
    }
}
```

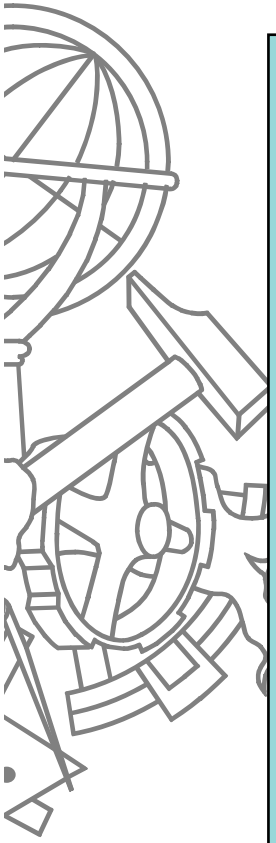
Aqui fazia sentido usar o padrão Strategy.
Ver Domain Model com Data Mapper

«get» Product



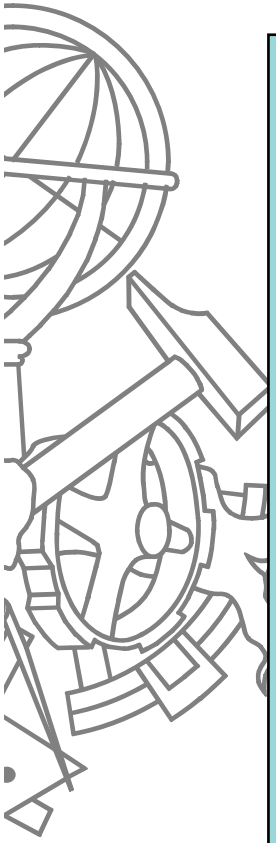
```
protected Product Product
{
    get
    {
        // Lazy Load
        if (this._Product == null)
          this._Product = Product.LoadById(this.ProductID);

        return this._Product;
    }
}
```

Product.LoadByID()



```
// Identity Map
public static IDictionary loaded = new Hashtable();

public static Product LoadById(int productID)
{
    // check registry - Identity Map
    Product p = (Product)loaded[productID];
    if (p != null)
        return p;

    // load
    Product aux = new Product();
    DataSet ds = aux.ExecuteQuery("SELECT * FROM TProducts WHERE productID=" +
    productID);
    p = new Product(ds.Tables[0].Rows[0]);

    // save in registry
    loaded[productID] = p;

    return p;
}
```

Save()



```
public void Save()
{
    BeginTransaction();
    object[] parms = new object[] {this.ProductID, this.CustomerID,
                                   this.DateSigned, this.Revenue, this.ID};
    if (this.ID != 0) {
          sqlContract = "UPDATE TContracts SET productId=?, customerID=?,
    dateSigned=?, revenue=? WHERE contractID=?";
          ExecuteTransactedNonQuery(sqlContract, parms);
          ExecuteTransactedNonQuery("DELETE FROM TRevenueRecognitions WHERE
    contractID=" + this.ID);
    } else {
          sqlContract = "INSERT INTO TContracts(productId, customerId,
    dateSigned, revenue) VALUES(?, ?, ?, ?)";
          this.myID = ExecuteTransactedNonQuery(sqlContract, parms);
    }

    foreach (RevenueRecognition r in _RevenueRecognitions)
    {
          string sqlRecognition = "INSERT INTO TRevenueRecognitions(contractID,
    dateRecognition, amount) VALUES(?, ?, ?)";
          object parms[] = new object[] {this.ID, r.DateRecognition, r.Amount};

          ExecuteTransactedNonQuery(sqlRecognition, parms);
    }
    CommitTransaction();
}
```