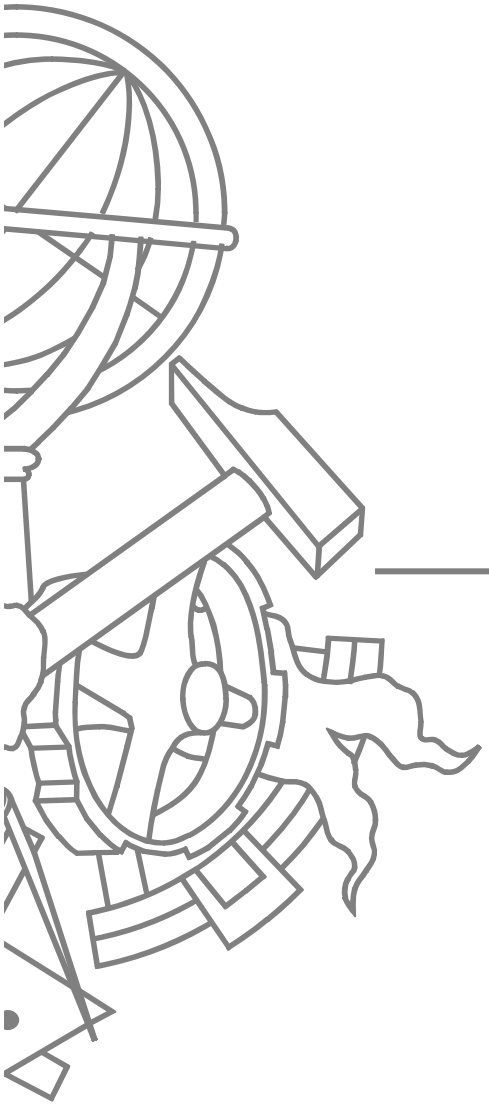
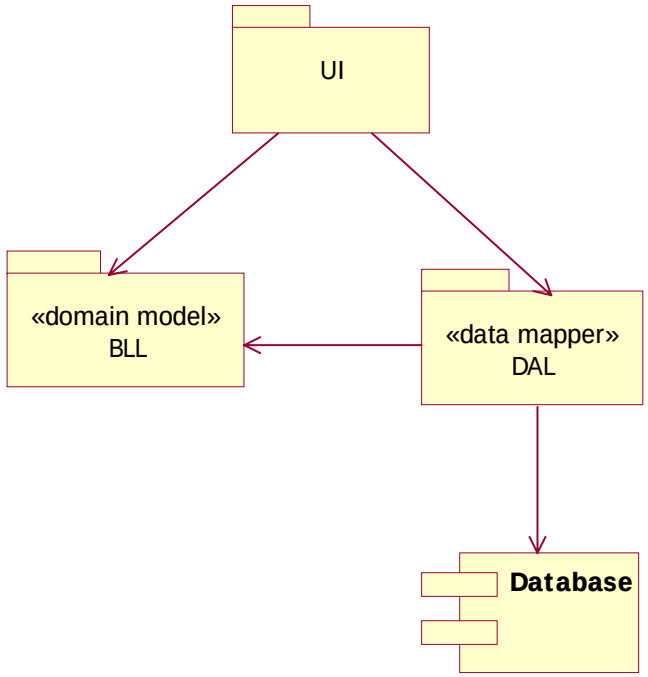
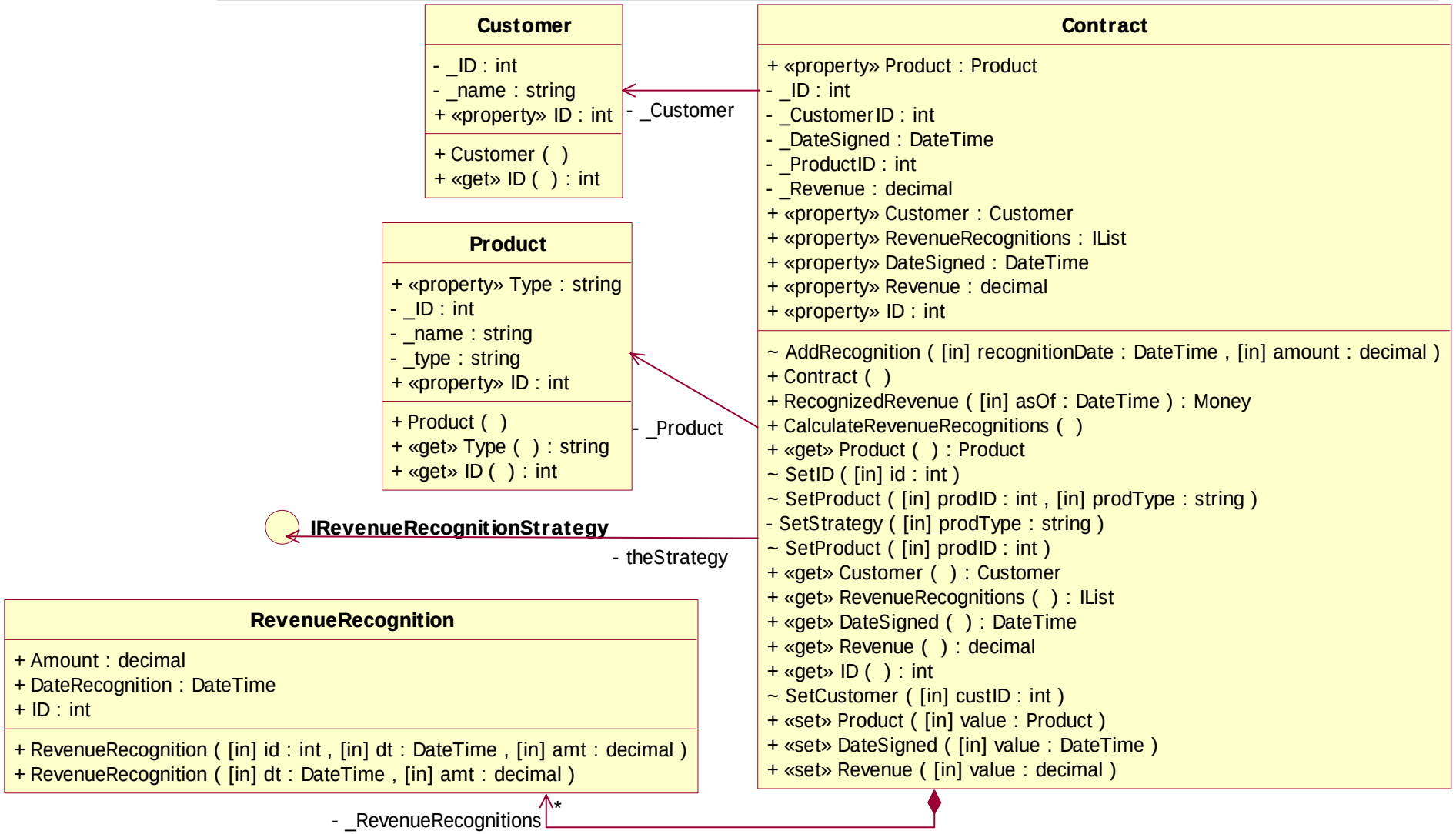


Domain Model com Data Mapper

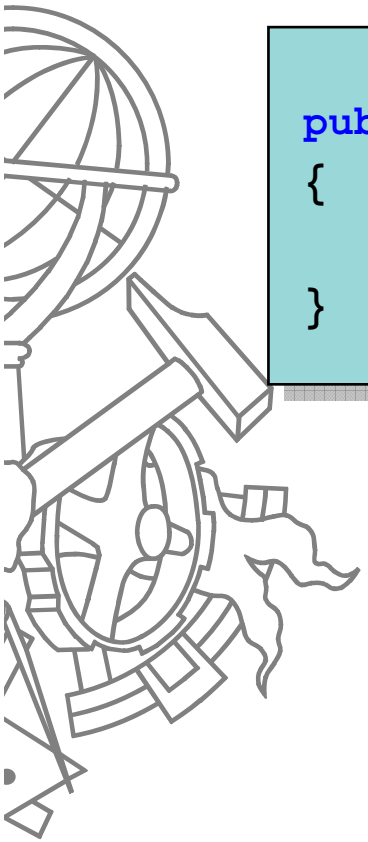
Packages



Classes BLL


Customer
- _ID : int
- _name : string
+ «property» ID : int
+ Customer ( )
+ «get» ID ( ) : int
- _Customer
Contract
+ «property» Product : Product
- _ID : int
- _CustomerID : int
- _DateSigned : DateTime
- _ProductID : int
- _Revenue : decimal
+ «property» Customer : Customer
+ «property» RevenueRecognitions : IList
+ «property» DateSigned : DateTime
+ «property» Revenue : decimal
+ «property» ID : int
~ AddRecognition ([in] recognitionDate : DateTime , [in] amount : decimal)
+ Contract ( )
+ RecognizedRevenue ([in] asOf : DateTime) : Money
+ CalculateRevenueRecognitions ( )
+ «get» Product ( ) : Product
~ SetID ([in] id : int)
~ SetProduct ([in] prodID : int , [in] prodType : string)
- SetStrategy ([in] prodType : string)
~ SetProduct ([in] prodID : int)
+ «get» Customer ( ) : Customer
+ «get» RevenueRecognitions ( ) : IList
+ «get» DateSigned ( ) : DateTime
+ «get» Revenue ( ) : decimal
+ «get» ID ( ) : int
~ SetCustomer ([in] custID : int)
+ «set» Product ([in] value : Product)
+ «set» DateSigned ([in] value : DateTime)
+ «set» Revenue ([in] value : decimal)
Product
+ «property» Type : string
- _ID : int
- _name : string
- _type : string
+ «property» ID : int
+ Product ( )
+ «get» Type ( ) : string
+ «get» ID ( ) : int
- _Product
IRevenueRecognitionStrategy
- theStrategy
RevenueRecognition
+ Amount : decimal
+ DateRecognition : DateTime
+ ID : int
+ RevenueRecognition ([in] id : int , [in] dt : DateTime , [in] amt : decimal)
+ RevenueRecognition ([in] dt : DateTime , [in] amt : decimal)
- _RevenueRecognitions
*

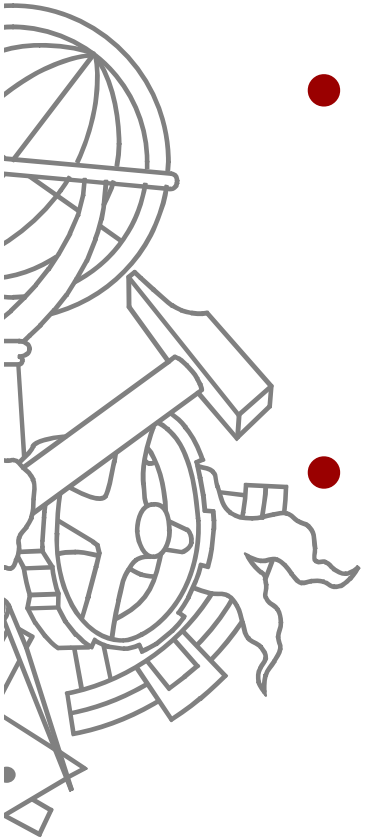
Interface IRevenueRecognitionStrategy



```
public interface IRevenueRecognitionStrategy
{
   void CalculateRevenueRecognitions();
}
```

Pattern GoF

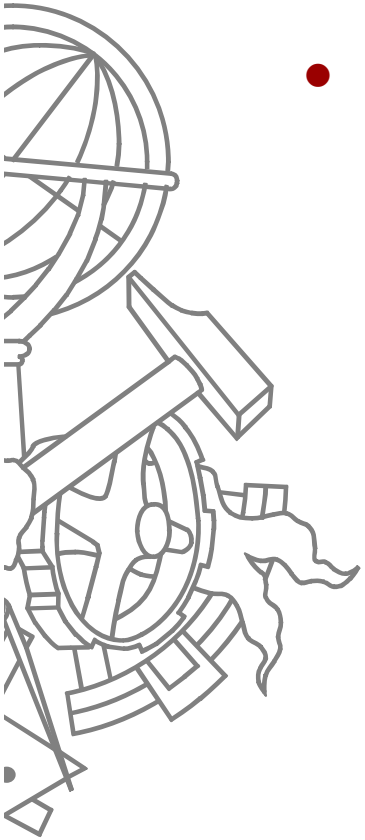
Strategy



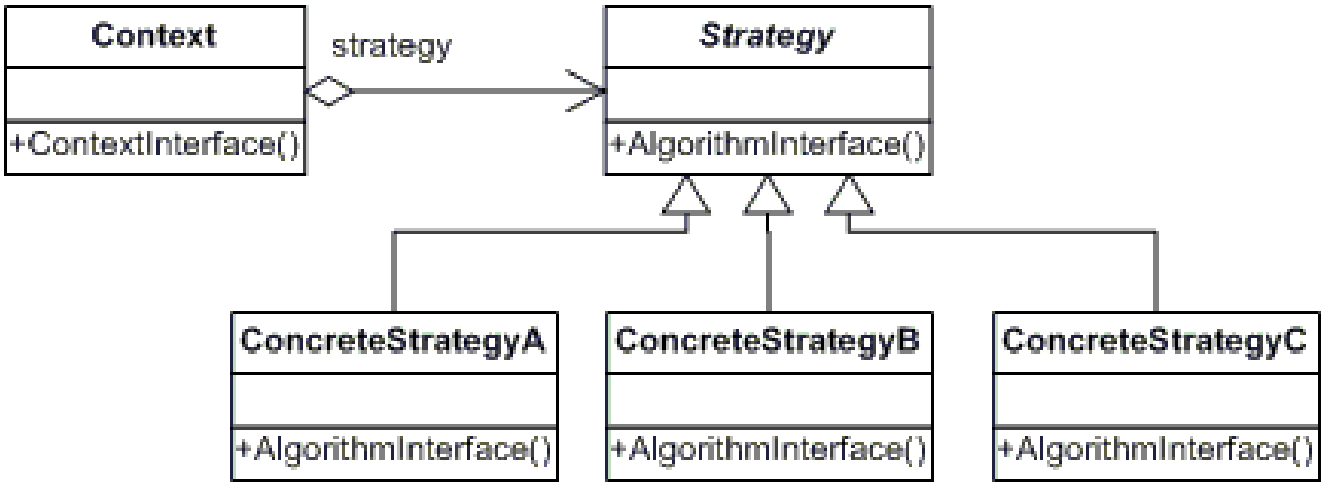
- **Problem:**
  - Allow the client the choice of many alternatives, but each is complex, and you don't want to include code for all.
- **Solution:**
  - Make many implementations of the same interface, and allow the client to select one and give it back to you.

Pattern GoF

Strategy

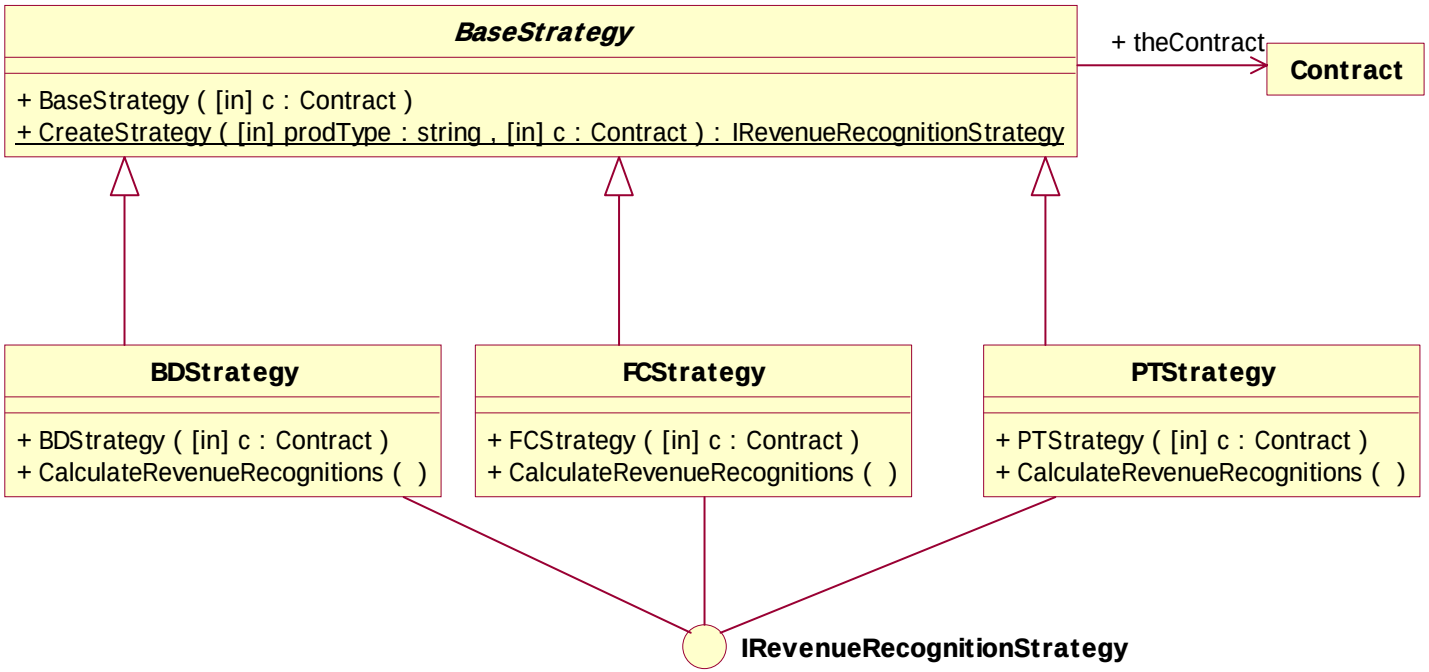


- Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

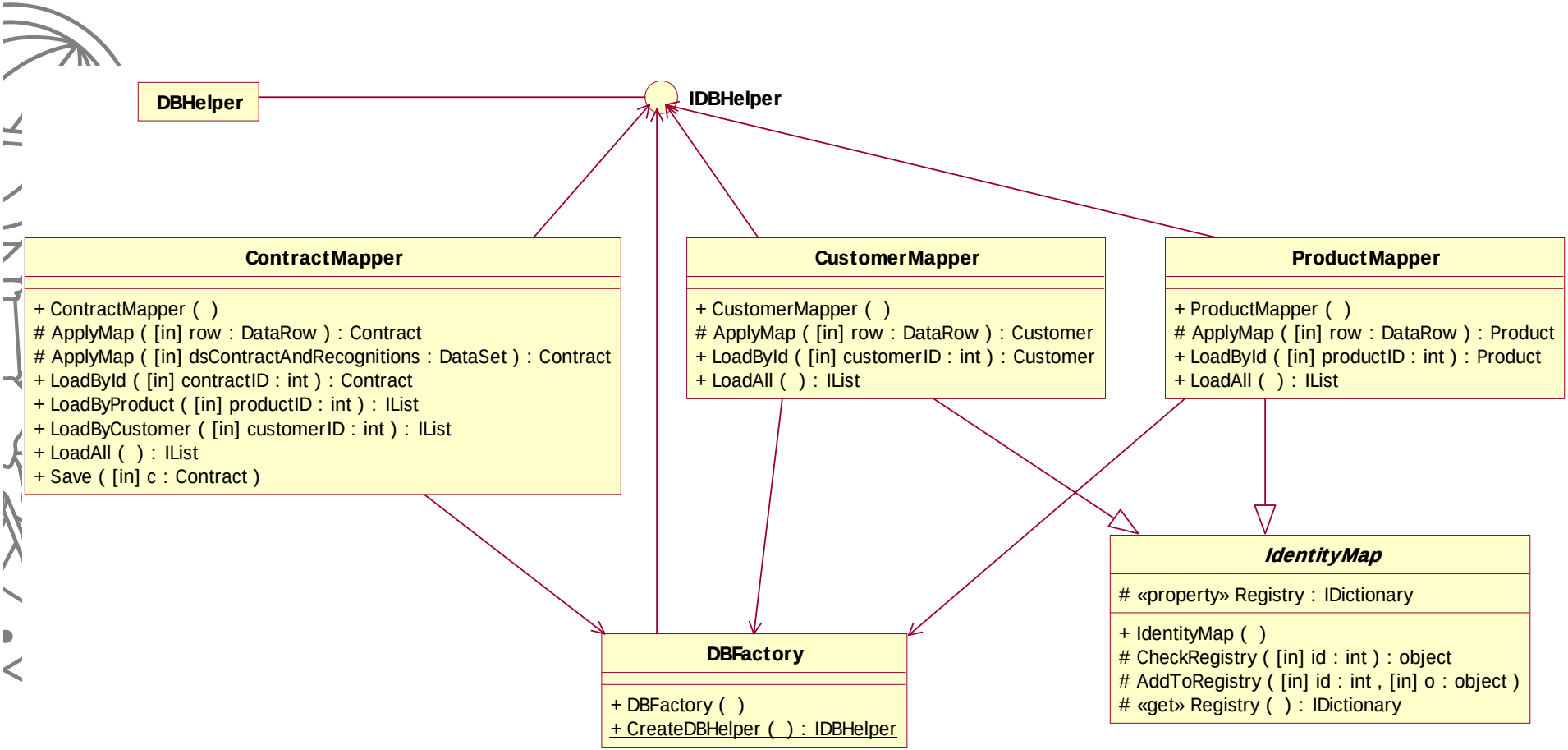


fonte: Design Patterns: Elements of Reusable Object-Oriented Software

Classes BLL (cont.)



Classes DAL


DBHelper
IDBHelper
ContractMapper
+ ContractMapper ( )
# ApplyMap ([in] row : DataRow) : Contract
# ApplyMap ([in] dsContractAndRecognitions : DataSet) : Contract
+ LoadById ([in] contractID : int) : Contract
+ LoadByProduct ([in] productID : int) : IList
+ LoadByCustomer ([in] customerID : int) : IList
+ LoadAll ( ) : IList
+ Save ([in] c : Contract)
CustomerMapper
+ CustomerMapper ( )
# ApplyMap ([in] row : DataRow) : Customer
+ LoadById ([in] customerID : int) : Customer
+ LoadAll ( ) : IList
ProductMapper
+ ProductMapper ( )
# ApplyMap ([in] row : DataRow) : Product
+ LoadById ([in] productID : int) : Product
+ LoadAll ( ) : IList
IdentityMap
«property» Registry : IDictionary
+ IdentityMap ( )
# CheckRegistry ([in] id : int) : object
# AddToRegistry ([in] id : int , [in] o : object)
# «get» Registry ( ) : IDictionary
DBFactory
+ DBFactory ( )
+ CreateDBHelper ( ) : IDBHelper

Classes DAL (cont.)

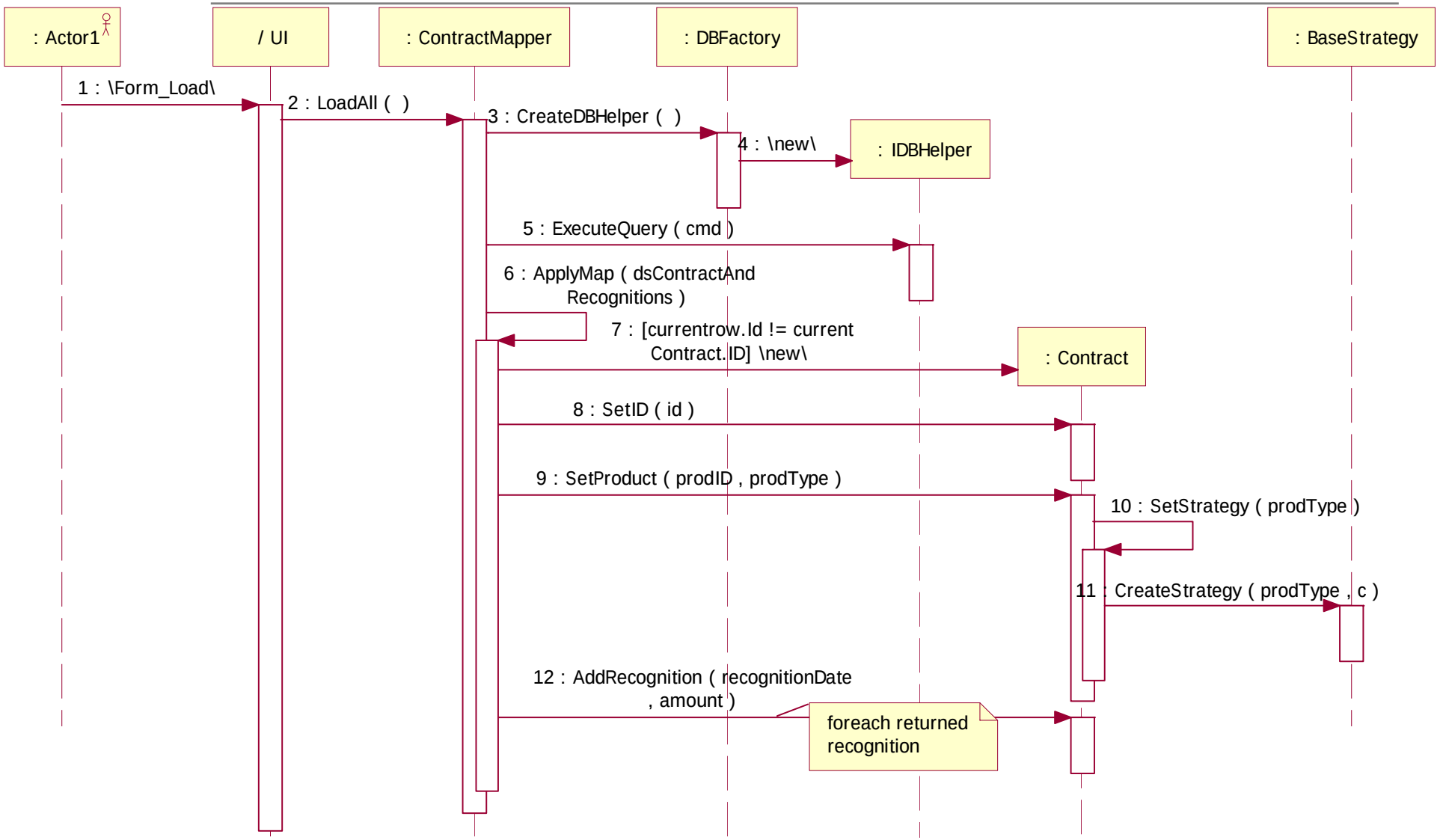
IDBHelper

DBHelper

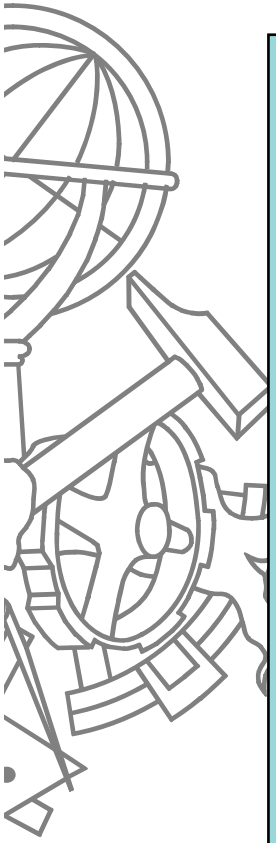
+ «property» CurrentTransaction : IDbTransaction
- CONNSTR : string = @"Provider=..."

+ DBHelper ( )
+ BeginTransaction ( )
+ CommitTransaction ( )
+ ExecuteNonQuery ([in] tx : IDbTransaction , [in] cmd : IDbCommand) : int
+ ExecuteNonQuery ([in] sql : string) : int
+ ExecuteQuery ([in] sql : string) : DataSet
+ ExecuteTransactedNonQuery ([in] sql : string) : int
+ ExecuteTransactedQuery ([in] sql : string) : DataSet
+ GetConnection ([in] open : bool) : IDbConnection
+ RoolbackTransaction ( )
+ «get» CurrentTransaction ( ) : IDbTransaction
+ ExecuteTransactedNonQuery ([in] cmd : IDbCommand) : int
+ CreateCommand ( ) : IDbCommand
+ CreateCommand ([in] inTransaction : bool) : IDbCommand
+ CreateParameter ([in] name : string , [in] value : object) : IDataParameter
+ CreateParameter ([in] name : string , [in] tp : DbType , [in] value : object) : IDataParameter
+ ExecuteQuery ([in] cmd : IDbCommand) : DataSet

Sequência GetContracts

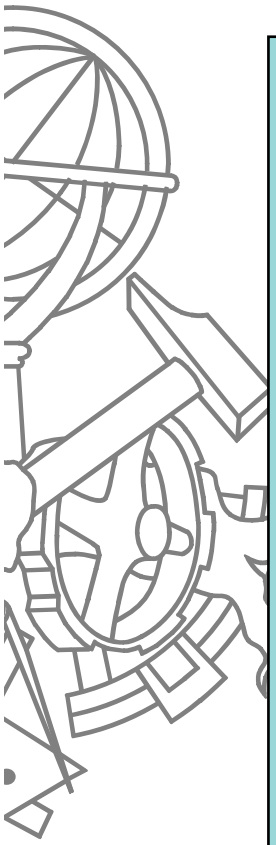


ApplyMap()



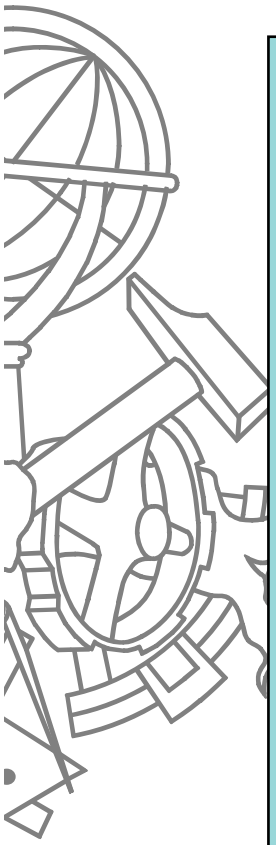
```
protected Contract ApplyMap(DataSet dsCNR)
{
    if (dsCNR].Rows.Count < 1)
          return null;

    Contract c = new Contract();

    c.DateSigned = (DateTime)dsCNR.Tables[0].Rows[0]["Datesigned"];
    c.Revenue = (decimal)dsCNR.Tables[0].Rows[0]["Revenue"];
    c.SetID( (int)dsCNR.Tables[0].Rows[0]["ID"] );
    c.SetProduct( (int) dsCNR.Tables[0].Rows[0]["ProductID"],
                  (string) dsCNR.Tables[0].Rows[0]["ProdType"]
          );
    c.SetCustomer( (int)dsCNR.Tables[0].Rows[0]["CustomerID"] );

    foreach (DataRow r in dsCNR.Tables[0].Rows) {
          c.AddRecognition( (DateTime)r["dateRecognition"],
                            (decimal)r["amount"]);
    }

    return c;
}
```

SetProduct()/SetStrategy()



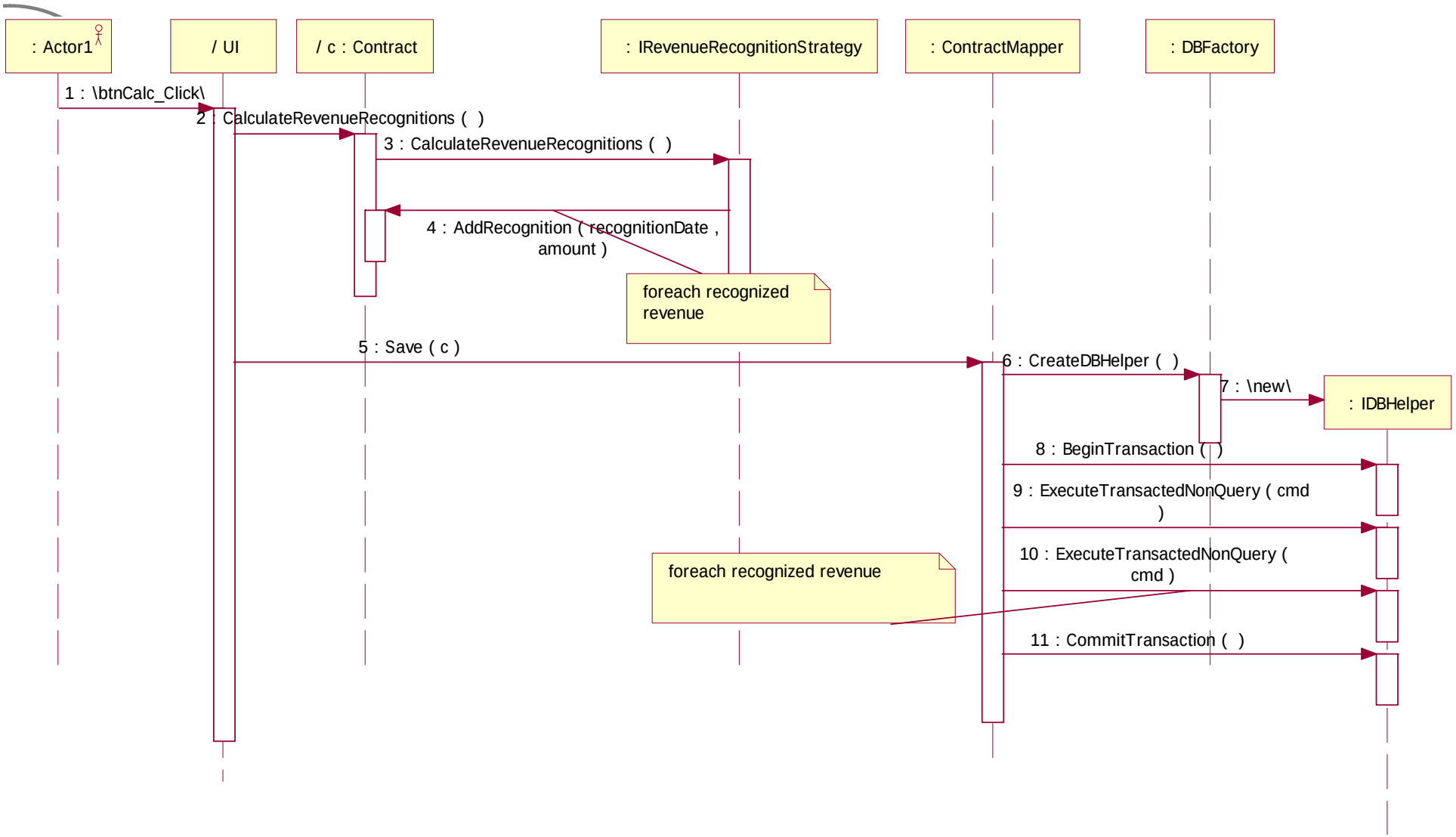
```
public class Contract
{
   // !!! to be used only by Mapper
   internal void SetProduct(int prodID, string prodType)
   {
        _ProductID = prodID;
        SetStrategy(prodType);
   }

   private void SetStrategy(string prodType)
   {
        theStrategy = BaseStrategy.CreateStrategy(prodType,
                                                  this);
   }
}
```

BaseStrategy

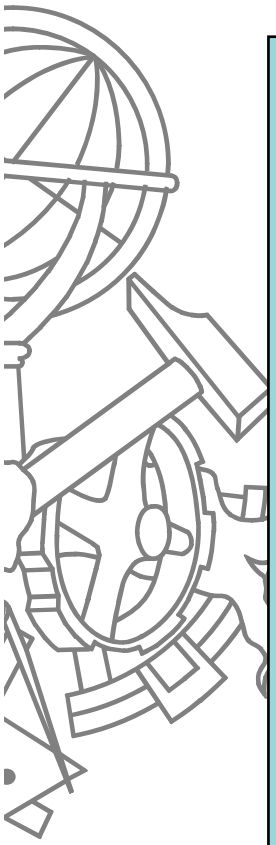


```
public abstract class BaseStrategy {
    protected Contract theContract;

    public BaseStrategy(Contract c) {
          theContract = c;
    }

    public static IRevenueRecognitionStrategy CreateStrategy(
                              string prodType, Contract c)
    {
          switch (prodType) {
          case "PT": return new Strategies.PTStrategy(c);
                    break;
          case "FC": return new Strategies.FCStrategy(c);
                    break;
          case "BD": return new Strategies.BDStrategy(c);
                    break;
          default:
                    throw new ApplicationException("invalid type");
          }
          return null;
    }
}
```

Sequência CalculateRevenues

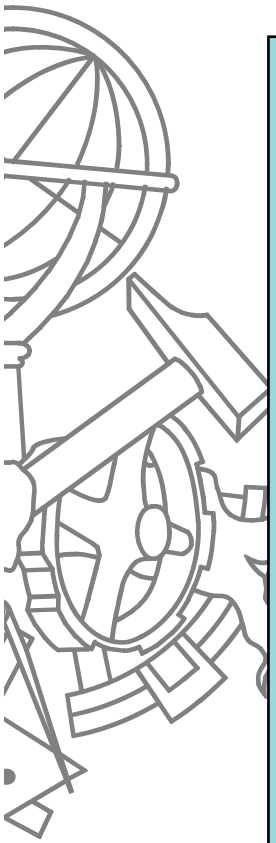


CalculateRevenues()



```
public void CalculateRevenueRecognitions()
{
    _RevenueRecognitions.Clear();
    theStrategy.CalculateRevenueRecognitions();
}

// !!! to be used by strategies
internal void AddRecognition(DateTime recognitionDate,
                             decimal amount)
{
    _RevenueRecognitions.Add(
         new RevenueRecognition(recognitionDate, amount)
         );
}
```

Strategy.CalculateRevenues()



```
public class BDStrategy : BaseStrategy {
    public void CalculateRevenueRecognitions()
    {
    decimal[] allocs = Money.Allocate(theContract.Revenue, 3);
    theContract.AddRecognition(theContract.DateSigned, allocs[0]);
    theContract.AddRecognition(theContract.DateSigned.AddDays(30), allocs[1]);
    theContract.AddRecognition(theContract.DateSigned.AddDays(60), allocs[2]);
    }
}

public class FCStrategy : BaseStrategy {
    public void CalculateRevenueRecognitions()
    {
    decimal[] allocs = Money.Allocate(theContract.Revenue, 3);
    theContract.AddRecognition(theContract.DateSigned, allocs[0]);
    theContract.AddRecognition(theContract.DateSigned.AddDays(60), allocs[1]);
    theContract.AddRecognition(theContract.DateSigned.AddDays(90), allocs[2]);
    }
}

public class PTStrategy : BaseStrategy {
    public void CalculateRevenueRecognitions()
    {
    theContract.AddRecognition(theContract.DateSigned, theContract.Revenue);
    }
}
```