Billing.Seed project

We have two projects already implemented:

- Billing.Database with all entities and database context defines
- Billing.Repository with Interface and Repository pattern implemented

Now, the project of data migration from “legacy data source” to “brand new database” will be explained.

Reasons for data migrations will not be explained here, just general workflow of actions. We will use the Excel file as a “legacy data source”. Each worksheet is one table with fields (columns) and records (rows). For accessing data we will use OleDb (Object Linking and Embedding, Database).

All common used variables, collections and methods are stored in *Help* class. All variables and methods are static, so we don’t need to instantiate – just use everything from *Help*.

Variables are:

```
// Database context using in repositories
public static BillingContext Context = new BillingContext();

// Path to Excel file on the disk - change due to your environment
public static string SourceRoot = @"C:\Billing\billing.xls";

// Set of dictionaries with key/value pairs - purpose will be explained below
public static Dictionary<int, int> dicAgen = new Dictionary<int, int>();
public static Dictionary<int, int> dicProd = new Dictionary<int, int>();
public static Dictionary<int, int> dicCatt = new Dictionary<int, int>();
public static Dictionary<int, int> dicShip = new Dictionary<int, int>();
public static Dictionary<int, int> dicSupp = new Dictionary<int, int>();
public static Dictionary<int, int> dicCust = new Dictionary<int, int>();
```

Method for getting data from worksheet and convert it to data table

```
// open worksheet named “sheet” and convert it to data table
public static DataTable OpenExcel(string sheet)
{
  // connection string
  var cs = string.Format("Provider=Microsoft.Jet.OLEDB.4.0;Data Source={0};Extended
                          Properties=Excel 8.0", SourceRoot);

  // declare and open connection to database (Excel file)
  OleDbConnection conn = new OleDbConnection(cs);
  conn.Open();

  // define command like select * from Products$
  OleDbCommand cmd = new OleDbCommand(string.Format("SELECT * FROM [{0}$]", sheet), conn);

  // create data adapter using defined command
  OleDbDataAdapter da = new OleDbDataAdapter(cmd);
  // define data table and fill it with data using data adapter
  DataTable dt = new DataTable();
  da.Fill(dt);
  // close connection
  conn.Close();

  // announce what table will be imported and return data table to a caller
  Console.Write($"{sheet}: ");
  return dt;
```

Having in mind database relations we have to import data to tables in well-defined order: we are going to fill data to "master" tables first, then data to "related" tables, to avoid "orphans/widows" errors. Our order of import will be (class Program, method Main) :

```
1.  Categories.Get();
2.  Products.Get();
3.  Towns.Get();
4.  Agents.Get();
5.  Shippers.Get();
6.  Suppliers.Get();
7.  Customers.Get();
8.  Procurements.Get();
9.  Invoices.Get();
10. Items.Get();
```

Products depend on Categories, so Categories have to be imported first. The same story stands for Towns vs Agents, Shippers, Suppliers and Customers. After imported all these data sets, we can import Procurements and Invoices. Finally we import Items, which are related to Invoices.

Defining domain classes using EFCF, we decided primary keys for all tables to be Identity type (integer, auto increment). For all "master" tables, during process of import, records are going to have new primary keys. But in related tables in legacy database (excel sheets), we have old values for primary keys. To put data in correct relations during process of import, we will use dictionaries. For each "master" table we will create dictionary with key/value entries like oldId/newId pairs. Let's see how it works in *Suppliers* table:

```
// prepare repositories for suppliers and towns
IBillingRepository<Supplier> customers = new BillingRepository<Supplier>(Help.Context);
IBillingRepository<Town> towns = new BillingRepository<Town>(Help.Context);

// get data table from sheet "Suppliers"
DataTable rawData = Help.OpenExcel("Suppliers");

// initialize record counter (just for info)
int N = 0;
foreach (DataRow row in rawData.Rows)
{
  // get and old SupplierId
  int oldId = Help.getInteger(row, 0);

  // get related Town object: get Zip code from column 'B' and search Towns data set
  string zip = Help.getString(row, 1);
  Town town = towns.Get().FirstOrDefault(x => x.Zip == zip);

  // create new object Supplier
  Supplier supp = new Supplier()
  {
    Name = Help.getString(row, 2),
    Address = Help.getString(row, 3),
    Town = town
  };

  // increase number of inserted records
  N++;
  // add supplier object to database and commit changes
  suppliers.Insert(supp);
  suppliers.Commit();
  // after inserting data, supp.Id has a new value of SupplierId
  // so we can insert new entry in dictionary dicSupp
  Help.dicSupp.Add(oldId, supp.Id);
}
// after insertion process, write number of records
Console.WriteLine(N);
```

When we have all those dictionaries full of pairs oldId / newId, we can insert transaction records like Procurements or Invoices. For procurements, the method will looks like:

```
// all repositories we need for import
IBillingRepository<Procurement> procurements = new BillingRepository<Procurement>(Help.Context);
IBillingRepository<Supplier> suppliers = new BillingRepository<Supplier>(Help.Context);
IBillingRepository<Product> products = new BillingRepository<Product>(Help.Context);

// get data table from sheet "Procurements"
DataTable rawData = Help.OpenExcel("Procurements");

// initialize record counter (just for info)
int N = 0;
foreach (DataRow row in rawData.Rows)
{
  // get related Supplier object using dicSupp dictionary
  Supplier supplier = suppliers.Get(Help.dicSupp[Help.getInteger(row, 2)]); //::: SEE EXPLANATION BELOW

  // get related Product object using dicProd dictionary
  Product product = products.Get(Help.dicProd[Help.getInteger(row, 3)]);

  // create new object Customer
  Procurement procurement = new Procurement()
  {
    Document = Help.getString(row, 0),
    Date = Help.getDate(row, 1),
    Supplier = supplier,
    Product = product,
    Quantity = Help.getInteger(row, 4),
    Price = Help.getDouble(row, 5)
  };

  // increase number of inserted records
  N++;
  // add customer object to database
  procurements.Insert(procurement);
}
// commit changes
procurements.Commit();

// after insertion process, write number of records
Console.WriteLine(N);
```

EXPLANATION:

```
Supplier supplier = suppliers.Get(Help.dicSupp[Help.getInteger(row, 2)]);
```

We have to recognize the Supplier. First, we will get his Id (the old one) from the column <C>:

```
int oldId = Help.getInteger(row, 2);
```

When we have an old supplier's Id, we can look to dictionary dicSupp, and find appropriate entry using index:

```
int newId = Help.dicSupp[oldId]);
```

This is new Id, the supplier got it when inserted in database in method Suppliers.Get(). Using this newId, we can go to repository and find appropriate supplier:

```
Supplier supplier = suppliers.Get(newId);
```

Of course, instead of writing three statemens and declaring extra variables, we can do this like written above ☺