

Relatório sobre desenvolvimento do projeto

“Plataforma Web para clientes da fabrica de calçado”

TECNOLOGIAS DA INTERNET II

1º ANO 2º SEMESTRE LICENCIATURA EM INFORMÁTICA

APRESENTADO POR:

SERGII SHTACHENKO

PEDRO SOARES

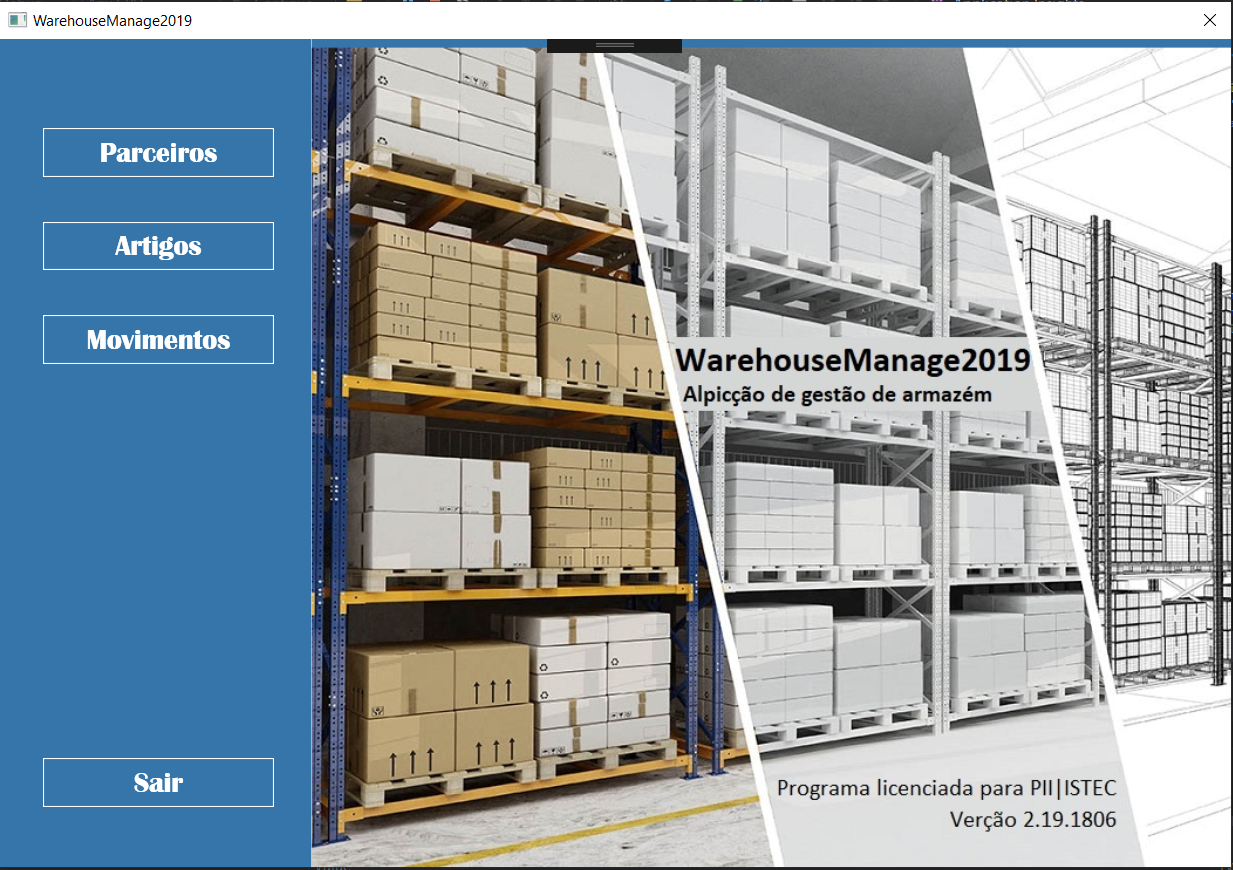
Porto, 26 de junho 2019

**Descrição**

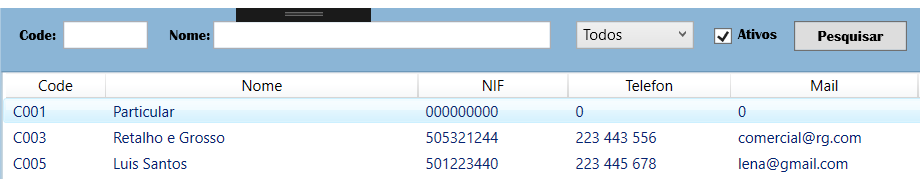
A aplicação está destinada para gestão de um armazém. Para isso a aplicação permite criar e gerir duas tabelas: Parceiros e Artigos. Os Parceiros são divididos em duas tipos: Clientes e Fornecedores.

Antes de fazer qualquer ação com o armazém esses tabelas bases têm que ser preenchidas.

O pragaram tem o seguinte menu principal:



Cada destes submenus representa a lista dos seus itens e permite procurá-los facilmente através de barra de pesquisa em sima da tabela.



Para diminuir a quantidade de consultas ao base de dados, a lista dos itens previamente guardada numa lista de respetivos objetos e ao clicar “Pesquisar” apresenta itens que correspondem aos valores procurados. Contudo, a consulta por defeito devolve apenas os itens com estado 1 (ativos). Ao desativar campo “Ativos” o programa vai fazer uma nova consulta ao base de dados para devolver itens com estado 0 (inativos).



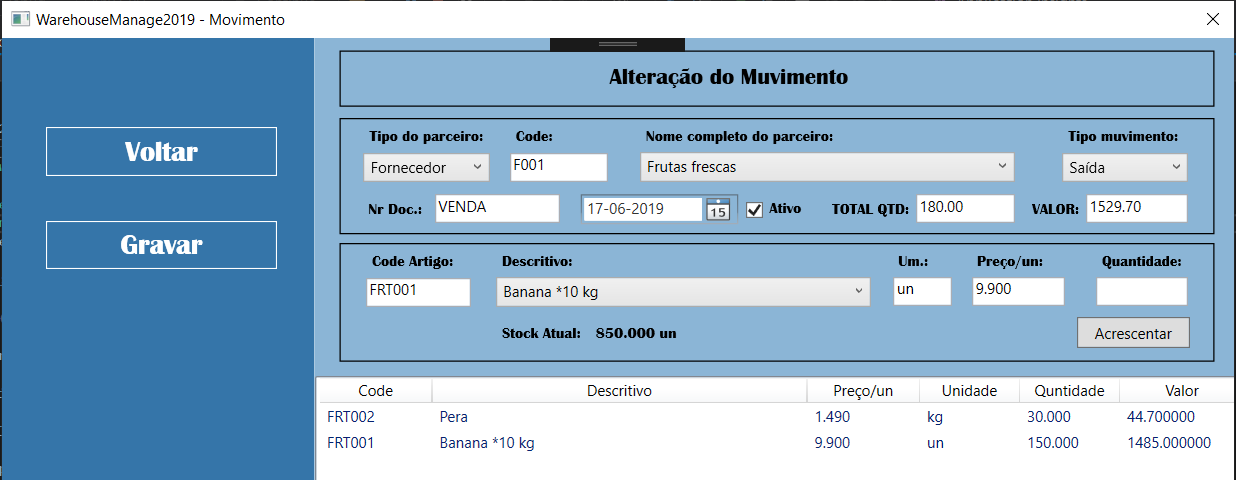
Cada dos submenus permite fazer as seguintes ações:

* Inserir novo item
* Visualizar os detalhes do item existente
* Alterar os detalhes do item incido inativá-lo caso seja necessário
* Apagar o item desnecessário (Atenção que impossível apagar o item que já tem um movimento associado.)

Toda essa funcionalidade implementada utilizando um único objeto de edição/visualização. Ao chamar esse objeto passamos para lá usando construtor o objeto de edição/ visualização e em que modo vamos abrir o objeto. Em modo de visualização o programa bloqueia a edição de todos os campos, pois é apenas para ver.

Uma parte do programa mais complexo é a criação dos movimentos, uma vez que numa janela de edição vamos utilizar todos as nossas classes: Parceiros, Movimento e Artigo.

Nesta janela de movimento temos que preencher os detalhes do documento e acrescentar (associar) vários Artigos existentes e Ativos.



Para garantir que criamos um movimento da saída ou da entrada de Artigos existentes e com os Parceiros existentes a lista de Parceiro e de Artigo é carregada de base de dados. Ao escolher Tipo de parceiro Fornecedor carrega-se a lista dos fornecedores aos escolher Cliente – clientes. Tipo de movimento muda-se astomático para Entrada caso for Fornecedor e Saída caso Cliente. Mas podemos alterar tipo de movimento para registar uma devolução. Campos Code, Total Qtd e Valor são te preenchimento automático.

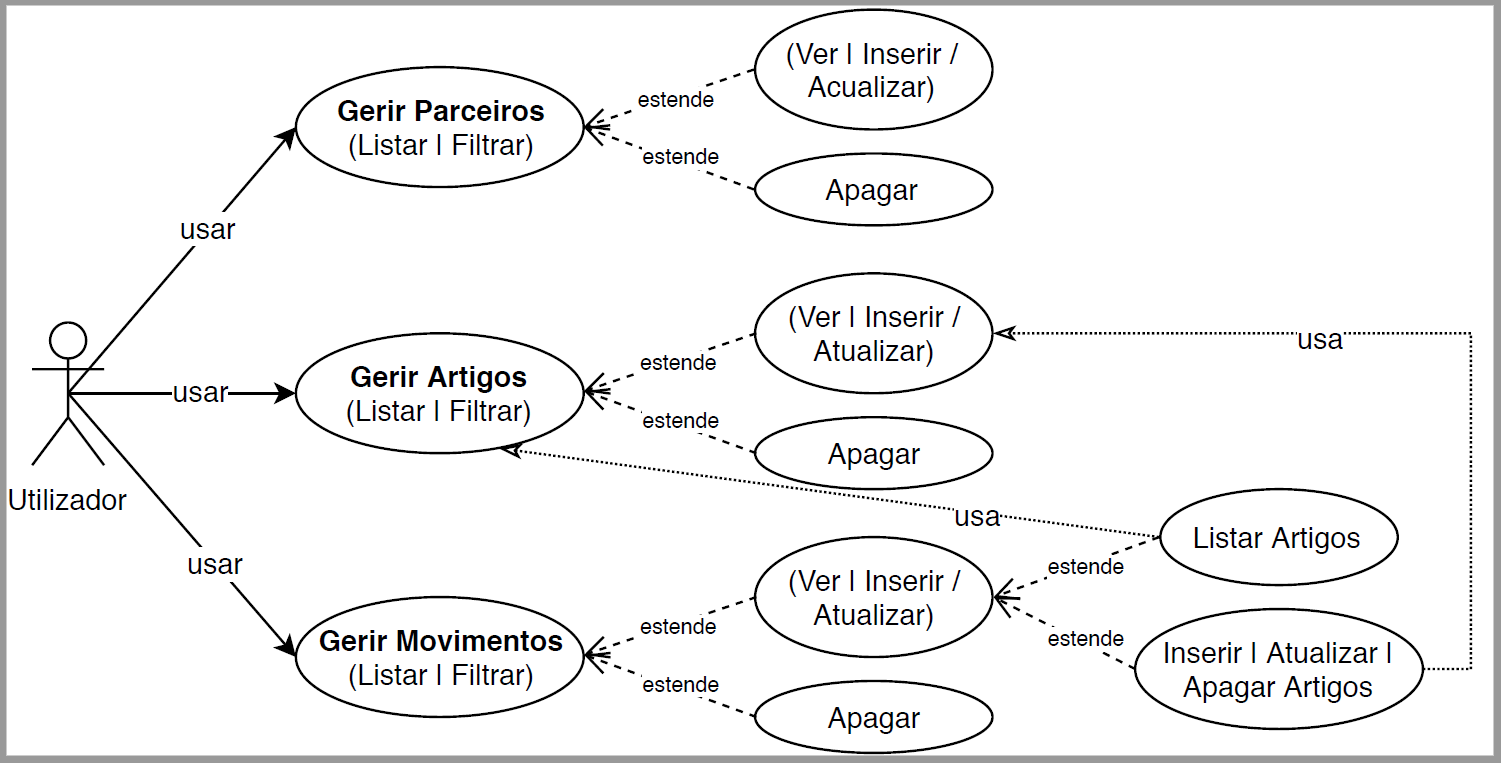
No caso da Saída de mercadoria o programa está a verificar o stock atual. Mesmo no caso de alteração de um movimento já existente o programa não deixa acrescentar artigos qua não têm o stock suficiente.

*É implementado com a formula: Stock do DB + Qtd dos itens guardados no registo – Qtd dos itens na ObservableCollection.*

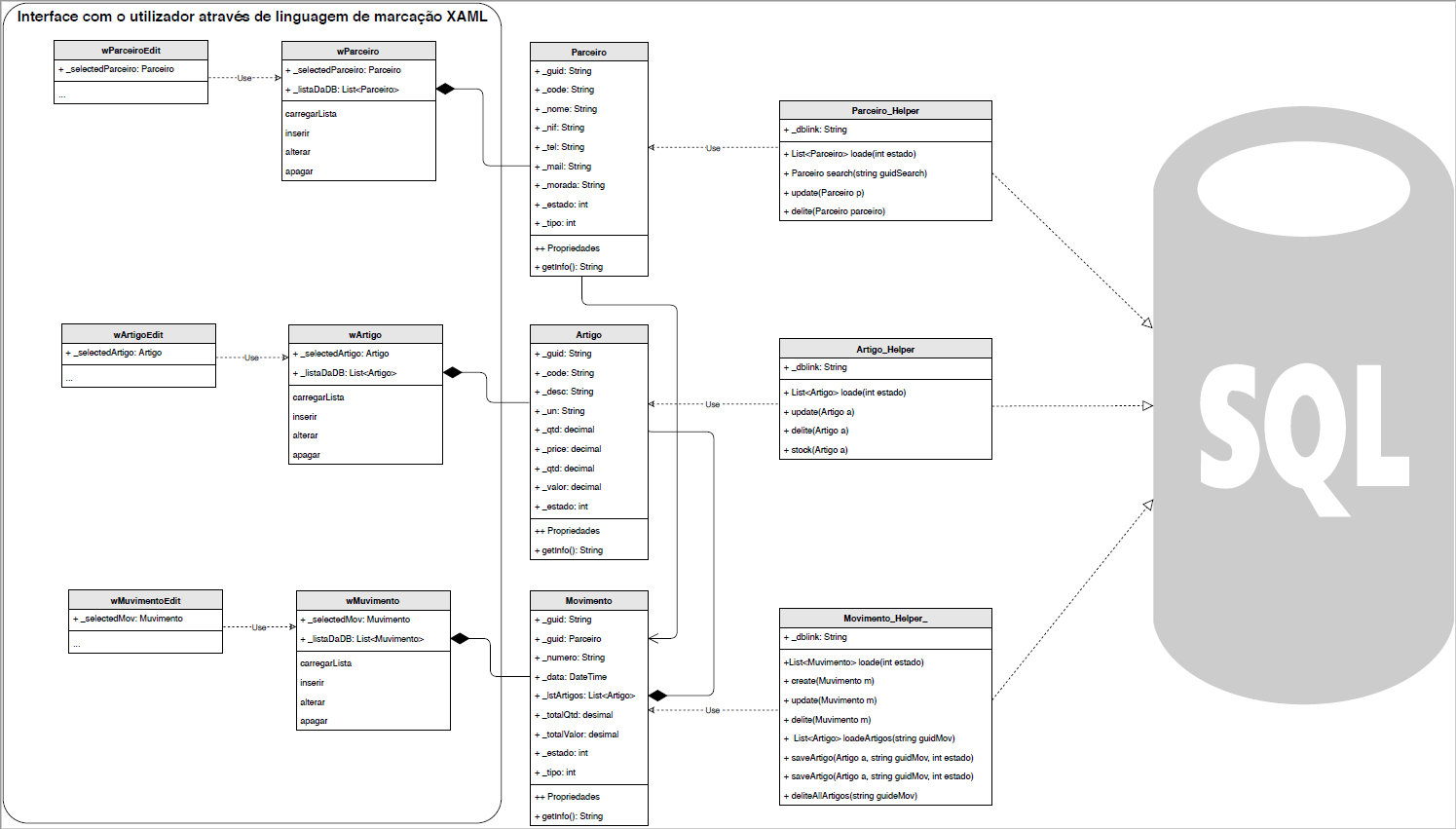
Os artigos da movimento estão guardados/alterados apenas após guardar o documento de movimento todo.

**Desenvolvimento do programa**

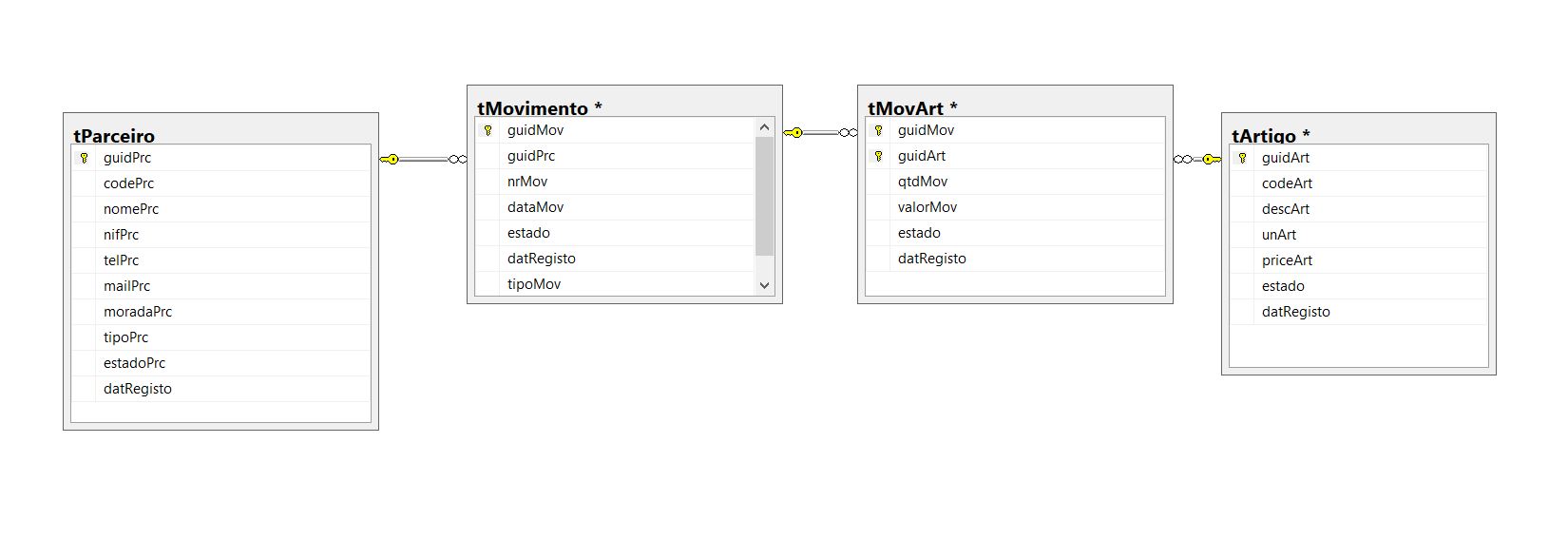
O programa foi desenvolvido em base de seguintes requisitos de funcionalidade:



Para implementar esses requisitos foram desenvolvido o seguinte arquitetura da programa desenhada numa tabela UML - diagrama de classes:



Do lado da Base de dados (DB) para armazenar os respetivos registos forma criados as seguintes tarbelas



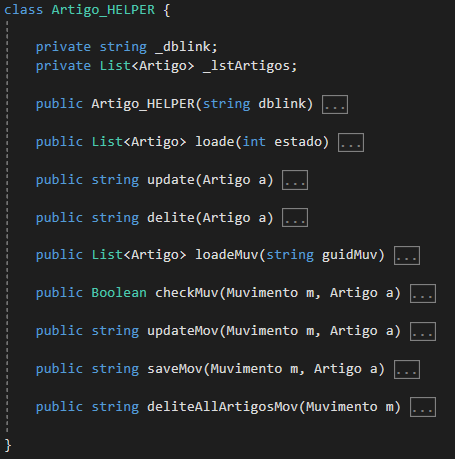
Tabelas tParceiro, tArtigo e tMovimento foram criados para guardar dados dos respetivos classes de aplicação. A Tabela tMovArt foi criada como um fruto de relação “muitos para muitos” de tMovimento com tArtigo. Guid da tabela tParceiro está armazenado como uma chave estrangeiro na tabela tMovimento que faz associar cada movimento com o respetivo parceiro.

O desenvolvimento deste projeto podemos dividir em duas partes:

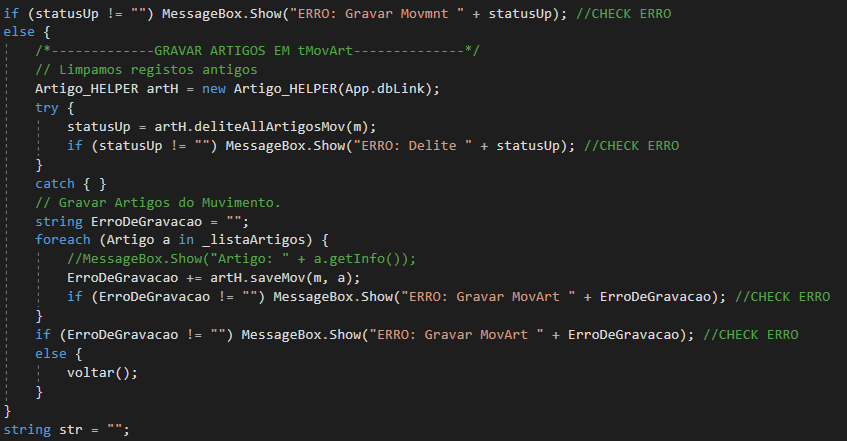
1. Primeira vamos chamar “Em cima de joelhos”
2. Segunda consideramos “Programação orientada a objetos”

**Desenvolvimento 0.1 – “Em cima de joelhos”**

Na primeira versão do programa foram criados 3 classe (Parceiro, Artigo e Muvimento) e os respetivos HELPERs. Cada HELPER possuiu os métodos de acesso e cação com a respetiva tabela na base de dados.

A classe Muvimento estava a guardar “guid” do Parceiro para guardar como chave estrangeira na DB. Uma vez que os atributos da classe Artigo levam a informação necessária para guardar nas tabelas quer tArtigo quer tMovArt, o Artigo\_HELPER possuiu também os métodos para agir com a tabela tMovArt (ver imagem).

Mas essa abordagem abriga a passar toda a funcionalidade de guardar os registos do movimento um a uma para a própria classe da janela wMuvimentoEdit. Portanto a programa torna-se pouco escalável e dificulta manutenção.



Alem disso no modo de visualizar/alterar temos que fazer mais uma consulta ao DB para devolver os dados do Parceiro associado ao nosso movimento.

Portanto surgiu uma necessidade de implementar uma nova versão 2.0 (Programação orientada a objetos)

**Desenvolvimento 2.0 – “Programação orientada a objetos”**

A nova versão já pode ser chamada Programação orientada a objetos. Pois foram implementadas as seguintes alterações.

Classe “Muvimento” possuiu novos atributos com respetivas propriedades:

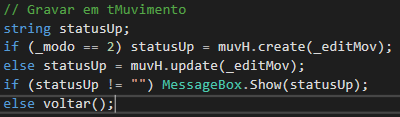
* Um objeto inteiro do tipo Parceiro
* Uma lista de objetos do tipo Artogo

Classe “Muvimento\_HELPER” possuiu os métodos privados para agir com a tabela tMovArt da DB:

* Carregar a lista dos artigos
* Guardar artigo de movimento
* Eliminar todos os artigos associados ao movimento

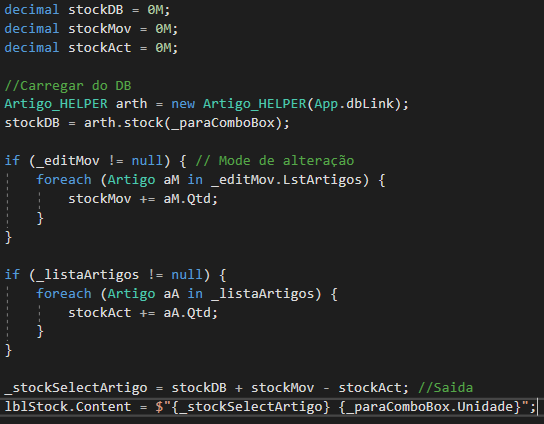
Esses métodos privados estão utilizados nos métodos já existentes da classe, ou seja, ao carregar a lista dos movimentos já carregamos com o respetivo Parceiro e a lista de todos os artigos associados. Na gravação de movimento HELPER vai gravar os dados do movimento, depois limpa os artigos associados a esse movimento de DB (no caso de alteração) e depois grava de novo os artigos da sua lista usando sucessivamente o método privado “saveArtigo”.

Desta forma a gravação de um novo movimento passou a um simples código:



Desta forma a programa tornou-se a ser facilmente escalável, pois é fácil criar nova janela e utilizar esse código quando precisa.

Outra vantagem de ter todos os artigos agregados num movimento permitiu criar nova funcionalidade importante – verificar stock disponível. Quando estamos a alterar um movimento existente temos que saber stock do DB, quantidade dos artigos do movimento original e quantidade do movimento novo:



Uma função (de facto propriedade) muito útil para conseguir copiar o Artigo para um movimento é:

