Министерство образования и науки Украины

НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИТЕТ

"ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ"

КАФЕДРА КОМПЬЮТЕРНОГО МОНИТОРИНГА И ЛОГИСТИКИ

КУРСОВАЯ РАБОТА

На тему: "Управление базой данных на языке программирования С+"

Выполнила студент гр. ЭИМ-39

Вычисенко М.А.

Руководитель: Клименко А.А.

Харьков 2011

Содержание

Введение

[1. Паттерны проектирования](#_Toc291876064)

1.1 Абстрактная фабрика

[1.2 Репозитории](#_Toc291876066)

2. Анализ предметной области

[3. Разработка пользовательского интерфейса](#_Toc291876068)

Заключение

[Список использованной литературы](#_Toc291876070)

# Введение

Объектно-ориентированное или объектное программирование (в дальнейшем ООП) - парадигма программирования, в которой основными концепциями являются понятия объектов и классов (либо, в менее известном варианте языков с прототипированием, - прототипов).

ООП возникло в результате развития идеологии процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны. Для дальнейшего развития объектно-ориентированного программирования часто большое значение имеют понятия события (так называемое событийно-ориентированное программирование) и компонента (компонентное программирование, КОП).

Формирование КОП от ООП произошло, как случилось формирование модульного от процедурного программирования: процедуры сформировались в модули - независимые части кода до уровня сборки программы, так объекты сформировались в компоненты - независимые части кода до уровня выполнения программы. Взаимодействие объектов происходит посредством сообщений. Результатом дальнейшего развития ООП, по-видимому, будет агентно-ориентированное программирование, где агенты - независимые части кода на уровне выполнения. Взаимодействие агентов происходит посредством изменения среды, в которой они находятся.

Первым языком программирования, в котором были предложены принципы объектной ориентированности, была Симула. В момент своего появления (в 1967 году), этот язык программирования предложил поистине революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Тем не менее, большинство концепций были развиты Аланом Кэйем и Дэном Ингаллсом в языке Smalltalk. Именно он стал первым широко распространённым объектно-ориентированным языком программирования.

база язык программирование паттерн

В настоящее время количество прикладных языков программирования (список языков), реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. В области системного программирования до сих пор применяется парадигма процедурного программирования, и общепринятым языком программирования является язык C. Хотя при взаимодействии системного и прикладного уровней операционных систем заметное влияние стали оказывать языки объектно-ориентированного программирования.

История возникновения и развития технологий баз данных может рассматриваться как в широком, так и в узком аспекте. В широком аспекте понятие истории баз данных обобщается до истории любых средств, с помощью которых человечество хранило и обрабатывало данные. В таком контексте упоминаются, например, средства учёта царской казны и налогов в древнем Шумере (4000 г. до н.э.), узелковая письменность инков - кипу, клинописи, содержащие документы Ассирийского царства и т.п. Следует помнить, что недостатком этого подхода является размывание понятия "база данных" и фактическое его слияние с понятиями "архив" и даже "письменность". История баз данных в узком аспекте рассматривает базы данных в традиционном (современном) понимании. Эта история начинается с 1955 г., когда появилось программируемое оборудование обработки записей. Программное обеспечение этого времени поддерживало модель обработки записей на основе файлов. Для хранения данных использовались перфокарты.

Оперативные сетевые базы данных появились в середине 1960-х. Операции над оперативными базами данных обрабатывались в интерактивном режиме с помощью терминалов. Простые индексно-последовательные организации записей быстро развились к более мощной модели записей, ориентированной на наборы. За руководство работой DBTG (Data Base Task Group), разработавшей стандартный язык определения данных и манипулирования данными, Чарльз Бахман получил Тьюринговскую премию. В это же время в сообществе баз данных COBOL была проработана концепция схем баз данных и концепция независимости данных.

Следующий важный этап связан с появлением в начале 1970-х реляционной модели данных, благодаря работам Эдгара Ф. Кодда. Работы Кодда открыли путь к тесной связи прикладной технологии баз данных с математикой и логикой. За свой вклад в теорию и практику Эдгар Ф. Кодд также получил премию Тьюринга. Сам термин database (база данных) появился в начале 1960-х гг., и был введён в употребление на симпозиумах, организованных фирмой SDC (System Development Corporation) в 1964 и 1965 гг. [8]

# 1. Паттерны проектирования

В разработке программного обеспечения, шаблон проектирования или паттерн (англ. design pattern) - повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

"Низкоуровневые" шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные. На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы. Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

# 1.1 Абстрактная фабрика

Абстрактная фабрика (англ. Abstract factory) - порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся наследующиеся от него классы, реализующие этот интерфейс.

*Цель*

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

*Плюсы*

* изолирует конкретные классы;
* упрощает замену семейств продуктов;
* гарантирует сочетаемость продуктов.

*Минусы*

* сложно добавить поддержку нового вида продуктов.

*Применимость*

Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты. Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения. Система должна конфигурироваться одним из семейств составляющих ее объектов. Требуется предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.

*Пример С+*

using System;

class MainApp

{

public static void Main ()

{

// Abstract factory #1

AbstractFactory factory1 = new ConcreteFactory1 ();

Client c1 = new Client (factory1);

c1.run ();

// Abstract factory #2

AbstractFactory factory2 = new ConcreteFactory2 ();

Client c2 = new Client (factory2);

c2.run ();

// Wait for user input

Console. Read ();

}

}

// "AbstractFactory"

abstract class AbstractFactory

{

public abstract AbstractProductA CreateProductA ();

public abstract AbstractProductB CreateProductB ();

}

// "ConcreteFactory1"

class ConcreteFactory1: AbstractFactory

{

public override AbstractProductA CreateProductA ()

{

return new ProductA1 ();

}

public override AbstractProductB CreateProductB ()

{

return new ProductB1 ();

}

}

// "ConcreteFactory2"

class ConcreteFactory2: AbstractFactory

{

public override AbstractProductA CreateProductA ()

{

return new ProductA2 ();

}

public override AbstractProductB CreateProductB ()

{

return new ProductB2 ();

}

}

// "AbstractProductA"

abstract class AbstractProductA

{

}

// "AbstractProductB"

abstract class AbstractProductB

{

public abstract void Interact (AbstractProductA a);

}

// "ProductA1"

class ProductA1: AbstractProductA

{

}

// "ProductB1"

class ProductB1: AbstractProductB

{

public override void Interact (AbstractProductA a)

{

Console. WriteLine (this. GetType (). Name +

" interacts with " + a. GetType (). Name);

}

}

// "ProductA2"

class ProductA2: AbstractProductA

{

}

// "ProductB2"

class ProductB2: AbstractProductB

{

public override void Interact (AbstractProductA a)

{

Console. WriteLine (this. GetType (). Name +

" interacts with " + a. GetType (). Name);

}

}

// "Client" - the interaction environment of the products

class Client

{

private AbstractProductA abstractProductA;

private AbstractProductB abstractProductB;

// Constructor

public Client (AbstractFactory factory)

{

abstractProductB = factory. CreateProductB ();

abstractProductA = factory. CreateProductA ();

}

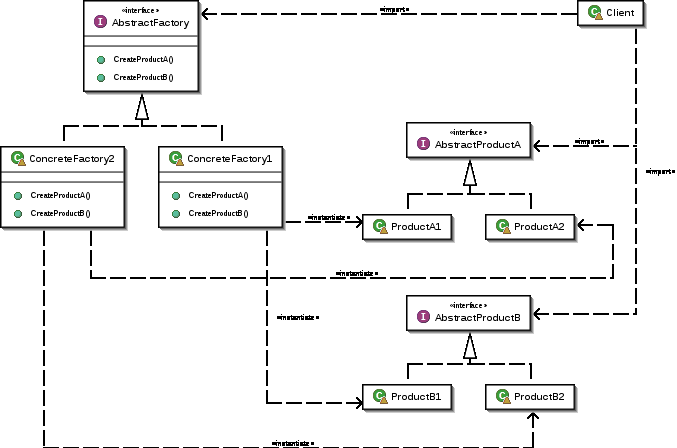
public void Run ()

{

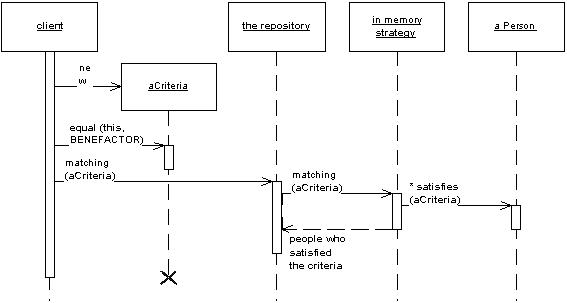
abstractProductB. Interact (abstractProductA);

}

}



# 1.2 Репозитории



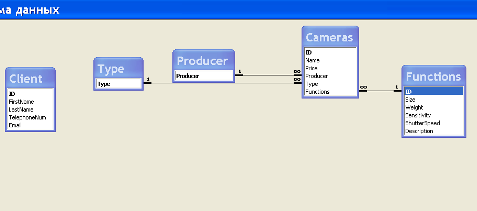
Посредничает между уровнями области определения и распределения данных (domain and data mapping layers), используя интерфейс, схожий с коллекциями для доступа к объектам области определения.

Система со сложной моделью области определения может быть упрощена при помощи дополнительного уровня, например Data Mapper, который бы изолировал объекты от кода доступа к БД. В таких системах может быть полезным добавление ещё одного слоя абстракции поверх слоя распределения данных (Data Mapper), в котором бы был собран код создания запросов. Это становится ещё более важным, когда в области определения множество классов или при сложных, тяжелых запросах. В таких случаях добавление этого уровня особенно помогает сократить дублирование кода запросов.

Паттерн Repository посредничает между слоем области определения и слоем распределения данных, работая, как обычная колекция объектов области определения. Объекты-клиенты создают описание запроса декларативно и направляют их к объекту-репозиторию (Repository) для обработки. Объекты могут быть добавлены или удалены из репозитория, как будто они формируют простую коллекцию объектов. А код распределения данных, скрытый в объекте Repository, позаботится о соответсвующих операциях в незаметно для разработчика. В двух словах, паттерн Repository инкапсулирует объекты, представленыые в хранилище данных и операции, производимые над ними, предоставляя более объектно-ориентированное представление реальных данных. Repository также преследует цель достижения полного разделения и односторонней зависимости между уровнями области определения и распределения данных.

# 2. Анализ предметной области

Предметной областью данной курсовой работы является магазин фотоаппаратуры с большим ассортиментом товаров. Для удобства работы "персонала" создана база данных в Microsoft Access 2003 с поддерживаемой целостностью данных.



В ней разработаны основные критерии, по которым оценивают и выбирают фотокамеры:

* цена;
* производитель;
* тип;
* габариты;
* вес;
* светочувствительность диафрагмы;
* скорость затвора/выдержка и другие.

Еще одним важным аспектом в разработке базы данных является клиентская база. С течением времени на основании этой базы можно будет разработать выгодную систему скидок или дисконтную систему, что увеличит доходы предприятия и привлечет новых клиентов.

# 3. Разработка пользовательского интерфейса

Сегодня ни для кого не секрет, что графический пользовательский интерфейс любого программного продукта является одним из ключевых факторов его популярности. Времена, когда связь между пользователем и приложением повсеместно устанавливалась при помощи командной строки, практически безвозвратно канули в лету, уступив место графическому пользовательскому интерфейсу (GUI от Graphical User Interface).

Создание грамотного пользовательского интерфейса - процесс трудоемкий и требующий максимального внимания к деталям. Создаваемый интерфейс должен максимально реализовывать возможности программы, но вместе с тем не перегружать пользователя обилием меню, кнопок, изображений и текста.

Даже самое мощное программное обеспечение, спроектированное талантливейшими инженерами и написанное самыми искусными программистами, без удобной организации взаимодействия с пользователем рискует так и остаться невостребованным.

Краеугольным камнем удобства программы либо интернет-сайта является его быстрота, которая достигается путем подробного изучения функций продукта и разработки максимально эффективного, интуитивно-понятного меню навигации, грамотным использованием иконок (ico), при котором пользователь не тратит время на поиски требуемой ссылки.

На практике программирование Windows-приложений предполагает экстенсивное использование различных инструментальных средств и мастеров, которые намного упрощают этот процесс Однако все указанные средства автоматизации заслоняют то, что лежит в основе создания графического пользовательского интерфейса Поэтому сначала мы рассмотрим основы создания графических пользовательских интерфейсов Иными словами, мы научимся создавать простые приложения Windows с самого начала, пользуясь только комплексом инструментальных средств разработки программ NET Framework SDK Это значит, что вначале мы будем создавать простые приложения Windows без применения каких-либо специальных сервисных программ Будут рассмотрены основы рисования с помощью Windows Forms (Формы Windows) с применением шрифтов и кистей, а также необходимые обработчики событий Мы объясним принципы обработки событий в Windows Forms (Формы Windows) и реализуем обработчики событий мыши.

С помощью Windows Forms (Формы Windows) мы также реализуем меню и соответствующие обработчики событий. Кроме того, мы рассмотрим управляющие элементы, а после этого изучим среду Visual Studio.net, посредством которой можно без труда создать простой графический пользовательский интерфейс на С#.

В нашем случае обязательным заданием являлась реализация трех кнопок (методов):

* добавление;
* изменение/обновление;
* удаление.

# Заключение

Эта программа может помочь многим людям, причем, не только покупателям, но и продавцам. Потому что очень часто люди покупают вовсе не то, что на самом деле искали или на что рассчитывали.

Это происходит из-за лени или оплошности продавцов, которые часто бывают некомпетентны. Но с этой программой они просто не смогут ошибиться, так как за них будет думать компьютер.

Не стоит упускать и тот факт, что в разработке программы использованы такие паттерны проектирования, как абстрактная фабрика и репозитории.

Это существенно облегчит задачу, если, допустим, в будущем магазин будет расширяться или клиентская база настолько вырастет, что одной базы данных станет недостаточно. Тогда её с легкостью можно будет заменить на СУБД, и работа программы не изменится.

# Список использованной литературы

1. http://donbass.ua/news/jobs-and-education/2010/10/13/prishla-pora-ukrupnjat-vuzy.html
2. [http://ru. wikipedia.org/wiki/Объектно-ориентированное\_программирование](http://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)
3. http://ru. wikipedia.org/wiki/Абстрактная\_фабрика\_ (шаблон\_проектирования)
4. [http://ru. wikipedia.org/wiki/Шаблон\_проектирования](http://ru.wikipedia.org/wiki/Шаблон_проектирования)
5. http://design-pattern.ru/patterns/repository.html
6. [http://www.db. by/services/interface/](http://www.db.by/services/interface/)
7. http://www.realcoding.net/article/view/2958