Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерный систем и сетей

Кафедра электронных вычислительных машин

Реферат на темы:

«Интерфейсы. Реализация нескольких интерфейсов с одним и тем же методом. Понятие Web-сервиса. Инфраструктура Web-сервиса. Взаимодействие клиентов и Web-сервисов»

Студент Иванов И.И.

Преподаватель Искра Н.А.

МИНСК 2011

СОДЕРЖАНИЕ

**1** ИНТЕРФЕЙСЫ 3

**1.1** Понятие Интерфейса 3

**1.2** Объявление Интерфейса3

**1.3** Реализация Интерфейса 4

**1.4** Реализация нескольких интерфейсов с одним и тем же методом5

**2** WEB-СЕРВИС 8

**2.1** Понятие Web-сервиса 8

**2.2** Инфраструктура Web-сервиса 9

**2.3** Взаимодействие клиентов и Web-сервисов 11

СПИСОК ЛИТЕРАТУРЫ13

**1** ИНТЕРФЕЙСЫ

**1.1** Понятие Интерфейса

Иногда в программировании полезно определить, что именно должен делать класс, но не как он должен это делать. Примером тому может служить абстрактный метод. В абстрактном методе определяются возвращаемый тип и сигнатура метода, но не предоставляется его реализация, а в производном классе должна быть обеспечена своя собственная реализация каждого абстрактного метода, определенного в его базовом классе. Так вот, интерфейс (англ. *interface*) представляет собой не более, чем просто набор каких-то абстрактных методов. В С# предусмотрено разделение интерфейса класса и его реализации с помощью ключевого слова interface.

С точки зрения синтаксиса, интерфейсы подобны абстрактным классам. Но в интерфейсе ни один из методов не должен быть реализован. В нем указывается только, что именно следует делать, но не как это делать. Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в конкретном классе может быть реализовано любое количество интерфейсов. [1]

 Для реализации интерфейса в классе должны быть предоставлены так называемые тела методов (т.е. конкретные реализации), описанных в этом интерфейсе. Каждому классу предоставляется полная свобода для определения деталей своей собственной реализации интерфейса. Это означает, что один и тот же интерфейс может быть реализован в двух классах по-разному.

Благодаря поддержке интерфейсов в С# может быть в полной мере реализован главный принцип полиморфизма: один интерфейс — множество методов. [2]

**1.2** Объявление Интерфейса

Интерфейсы объявляются с помощью ключевого слова interface. Приведем пример упрощенной формы объявления интерфейса:

interface имяИнтерфейса

{

возвращаемыйТип имяМетода1 (списокПараметров);

возвращаемыйТип имяМетода2 (списокПараметров);

/\* ... \*/

возвращаемыйТип имяМетодаN (списокПараметров);

}

где имяИнтерфейса — это конкретное имя интерфейса (необязательно, но имя интерфейса желательно начинать с I (например, IPlayer)). [1]

В объявлении методов интерфейса используются только их возвращаемыйТип и сигнатура. Как отмечалось выше, методы интерфейса, по сути, являются абстрактными методами и в интерфейсе не может быть никакой реализации (все методы интерфейса должны быть реализованы в каждом классе, включающем в себя этот интерфейс). В самом же интерфейсе методы неявно считаются открытыми, поэтому доступ к ним не нужно указывать явно.

Приведем пример объявления интерфейса, в котором дадим описание метода, характеризующее действие $–$ игры на чем-либо или с чем-либо:

interface IPlayer

{

void play(); // метод, реализующий игру

}

**1.3** Реализация Интерфейса

Как только интерфейс будет определен, он может быть реализован в одном или нескольких классах. Для реализации интерфейса достаточно указать его имя после имени класса (как при наследовании). Ниже приведена общая форма реализации интерфейса в классе:

class имяКласса : имяИнтерфейса

{

// тело класса

}

где имяИнтерфейса — это имя реализуемого интерфейса. Так же интерфейс можно реализовать в самом классе. В частности, реализовать интерфейс выборочно и только по частям нельзя.

В классе можно реализовывать сколь угодно интерфейсов. В этом случае все реализуемые в классе интерфейсы указываются списком через запятую. В классе можно наследовать базовый класс и в тоже время реализовать один или более интерфейс. Имя базового класса должно быть указано перед списком интерфейсов, разделяемых запятой. [1]

Еще важный момент $-$ методы, реализующие интерфейс, должны быть объявлены как public. Дело в том, что в самом интерфейсе эти методы неявно подразумеваются как открытые, поэтому их реализация также должна быть открытой. Кроме того, возвращаемый тип и сигнатура реализуемого метода должны точно соответствовать возвращаемому типу и сигнатуре, указанным в определении интерфейса. [3]

Ниже приведен пример программы, в которой реализуется представленный ранее интерфейс IPlayer. В данной программе создаются два класса – первый, для играющего на гитаре, второй – для кого-то, играющего с мячом.

public class Gitara: IPlayer

{

public void play(){

Console.WriteLine("Игра на гитаре");

}

}

public class Ball: IPlayer

{

public void play()

{

Console.WriteLine("Игра c мячом");

}

}

Как видно из кода, мы создаем два класса, которые реализуют один и тот же интерфейс IPlayer.

using System;

namespace Interface\_Example

{

class Program

{

public static void Main(string[] args)

{

Gitara playGitara = new Gitara();

playGitara.play();

Ball playBall = new Ball();

playBall.play();

}

}

}

Результатом выполнения программы будет вывод двух строк на экран: playGitara.play() выведет на экран строку Игра на гитаре, а playBall.play() – строку Игра c мячом. В итоге однотипное действие (играть) в объектах разных классов вызывается одним и тем же способом.

**1.4** Реализация нескольких интерфейсов с одним и тем же методом

Иногда может возникнуть ситуация, когда нам нужно реализовать два (или более) интерфейса с одним и тем же названием метода и теми же типами и количеством аргументов и даже тем же типом возвращаемого значения, и мы должны реализовать оба интерфейса в одном классе. Встает вопрос: как же реализовать интерфейсы в нашем классе?

Можно подумать, что ничего сложного в этом нет $–$ нам просто нужно реализовать интерфейс и метод в каждом интерфейсе. Если сигнатуры методов в интерфейсах разные, то не было бы никакой проблемы, но здесь сигнатуры методов в двух различных интерфейсах одинаковы, и оба интерфейса необходимо реализовать в одном классе. [3]

Приведем пример такой реализации:

public interface IA

{

string PrintName();

}

рublic interface IB

{

string PrintName();

 }

Из приведенного выше кода, мы можем заключить, что у нас есть два интерфейса с именами IA и IB, и оба имеют один метод под названием PrintName. Сигнатура в обоих методах одинакова.

Нам необходимо реализовать интерфейсы в нашем классе (назовем его класс А). Один из способов реализации интерфейса показан ниже - мы реализуем метод только один раз с модификатором public.

public class A : IA, IB

{

public A()

{

}

public string PrintName()

{

return “PrintName()”;

}

}

Такая реализация имеет свои ограничения. Методом PrintName считается общим методом для класса, и для интерфейсов IA и IB. Если вы напишете код, приведенный ниже

A a = new A();

IA ia = new A();

IB ib = new A();

Console.WriteLine(a.PrintName());

Console.WriteLine(ia.PrintName());

Console.WriteLine(ib.PrintName());

все вызовы метода PrintName дадут нам тот же результат $–$ на экран выведется строка PrintName(). Произойдет это потому, что все вызовы метода приводят к одному и тому же определению. Но нам необходимы различные реализации для методов в интерфейсах IA и IB. Сделать это достаточно просто. Есть две реализации одного и того же метода и префикс имен методов с именем интерфейса:

public class A : IA, IB

{

string IA.PrintName()

{

return “IA PrintName()”;

}

string IB.PrintName()

{

return “IB PrintName()”;

}

}

Теперь этот код даст нам другой результат

IA ia = new A();

IB ib = new A();

Console.WriteLine(ia.PrintName());

Console.WriteLine(ib.PrintName());

Теперь выводы строк на экран будут различаться: Console.WriteLine(ia.PrintName()) выведет строку IA PrintName(), а Console.WriteLine(ib.PrintName()) $–$ строку IB PrintName().

**2** WEB-СЕРВИС

**2.1** Предпосылки возникновения Web-сервиса

За последние годы Интернет необратимо повлиял на разработку приложений намного больше, чем любые другие силы. Практически любая организация все больше зависит от цифровых ресурсов, предоставляемых Интернетом и связанными с ним технологиями. Как итог $-$ сегодня лишь очень малая часть приложений проектируется и разрабатывается без учета оптимального использования технологий Интернет.

Если не преследуется цель разработки полностью нового приложения, организации пытаются создать приложения, объединяющие несколько традиционных приложений в одном составном приложении. Но тут возникает очень сложные проблемы при попытке объединить приложения, созданные с использованием разнообразных технологий, объектных моделей, на различных операционных системам и языках программирования. Как заставить их работать друг с другом? Ответ — программируемый Интернет!

Язык XML, как открытый формат описания данных, положил начало реальному программируемому Интернету. Подобно тому, протокол HTML стал стандартным языком отображения информации в Интернете, XML обеспечивает стандартный язык обмена данными для автоматической обработки. Он предоставляет возможность представления данных в широко поддерживаемом формате, позволяющем компьютерам отправлять и получать данные в интуитивно понятном и предсказуемом стиле. Язык XML дает свободу, так как его простота и расширяемость позволяют определить практически что угодно, сохраняя возможность расширения. Одним из главных строительных блоков программируемого Интернета являются XML-Web-сервисы или просто Web-сервисы (англ. *Web-service*). [5]

**2.1** Понятие Web-сервиса

Как такового, официального определения Web-сервиса нет, но основываясь на его задачах, можно заключить, что Web-сервис $-$ это программируемое средство предоставления определенного элемента или функциональных возможностей, доступное любому количеству потенциально различных систем с помощью широко применяемых стандартов Интернета, таких как XML и HTTP. [5]

Другими словами Web-сервис $-$ это код, доступный по протоколу HTTP и возвращающий информацию в формате XML конкретному приложению. Идея создания Web-сервисов, решающих многие стандартные задачи, очень привлекательна $-$ в этом случае мы получаем программируемые Web-приложения, которые могут располагаться где угодно и работать на любых платформах. Нужно отметить, что определение Web-сервиса, так же включает поддержку протокола SOAP (Simple Object Access Protocol или Простой Протокол Доступа к Объектам $-$ протокол обмена сообщениями на базе XML, который необходим, для реализации возможностей обратного вызова), а также протоколов WSDL (Web-Service Description Laguage или Язык Определения Web-сервисов $-$ язык описания внешних интерфейсов веб-службы на базе XML) и UDDI (Universal Description, Discovery, Interoperability или Универсальное Описание, Обнаружение, Взаимодействие $-$ каталог веб-служб и сведений о компаниях, предоставляющих веб-службы во всеобщее пользование или конкретным компаниям).

 Web-сервис можно использовать для одного приложения на компьютере или предоставить к нему доступ через Интернет любому числу приложений. Поскольку доступ к Web-сервису выполняется через стандартный интерфейс, с нем могут работать различные системы, образуя единую вычислительную сеть. [4]

**2.2** Инфраструктура Web-сервиса

По определению Web-сервиса он не должен зависеть от выбора операционной системы, объектной модели и языка программирования. Кроме того, чтобы Web-сервис принес пользу, как другие веб-технологии, он должен обладать определенными характеристиками.

* Слабая связанность: две системы считаются слабосвязанными, если единственным предъявляемым к ним требованием является передача информация входных и выходных данных и местоположения. Сильносвязанные системы, с другой стороны, требуют значительного объема настраиваемых служебных данных, чтобы обеспечить возможность взаимодействия, и большего взаимопонимания между системами.
* Взаимодействие в глобальном масштабе: способность подключить почти любую систему или устройство к Интернету обеспечивает таким системам и устройствам доступность со стороны любой другой системы или устройства, подключенного к Интернету в любой точке мира.
* Универсальный формат данных: использование существующих открытых стандартов (язык XML), а не фирменных методов взаимодействия в замкнутой среде; любая система, поддерживающая такие же открытые стандарты, способна понимать Web-сервисы.

Web-сервисы используют инфраструктуру, обеспечивающую механизм обнаружения (для поиска Web-сервисов), описание служб (для определения принципов их использования) и стандартные форматы подключения, с помощью которых должно осуществляться взаимодействие. На рисунке 2.1 приведен пример такой инфраструктуры:



Рисунок 2.1 $-$ Инфраструктура Web-сервиса

Компоненты инфраструктуры:

* Каталоги Web-сервисов: каталоги Web-сервисов представляют собой центральное расположение для размещения Web-сервисов (например: http://uddi.microsoft.org), предоставляемых другими организациями. Эту роль выполняют такие каталоги Web-сервисов, как реестр UDDI. Клиентам Web-сервиса может быть разрешено или не разрешено обращаться к каталогу Web-сервисов.
* Обнаружение Web-сервиса: процесс поиска или обнаружения одного или нескольких связанных документов, описывающих определенный Web-сервиса на языке WSDL. Алгоритм поиска описаний служб определен в спецификации DISCO (файловый механизм поиска локальных web-сервисов (от англ. *discovery*)). Если клиенты Web-сервиса знают расположение описания службы, они могут игнорировать процесс обнаружения.
* Описание Web-сервиса: чтобы определить, как взаимодействовать с конкретным Web-сервисом, необходимо обеспечить описание службы, в котором определяется, какие взаимодействия поддерживает этот Web-сервис. Прежде чем использовать Web-сервис, клиенты Web-сервиса должны выяснить, как взаимодействовать с ней.
* Форматы подключения Web-сервиса: чтобы обеспечить возможность универсального взаимодействия, Web-сервис взаимодействуют с использованием протоколов и поддерживает наиболее распространенные стандарты Интернета. Основным протоколом взаимодействия Web-сервиса является протокол SOAP. [5]

**2.3** Взаимодействие клиентов и Web-сервисов

Как ранее говорилось, стандарты Web-сервисов состоят из трех основных частей: WSDL, UDDI и SOAP. Их применение показано на рисунке 2.2:



Рисунок 2.2 $-$ Взаимодействие составных частей технологии Web-сервисов

Как видно из рисунка, кроме сервиса и его клиента в процессе применения Web-сервисов участвует каталог. Ранее упоминалось, что каталог $–$ это некоторое общедоступное хранилище описаний Web-сервисов. Каталоги поддерживаются фирмами и организациями, добровольно взявшими на себя эту функцию. Наиболее развитыми и общеизвестными являются каталоги фирм IBM и Microsoft. Разработчик, желающий сделать свой сервис общедоступным, составляет описание своего сервиса на языке WSDL. Это описание он помещает в каталог (публикует). Взаимодействие владельца сервиса с реестром происходит в соответствии со стандартом UDDI. Клиент, также используя UDDI, запрашивает в каталоге поиск Web-сервисов с нужными ему функциями и получает описания Web-сервисов, удовлетворяющие параметрам его запроса. Выбрав подходящий для него Web-сервис, клиент обращается к нему по протоколу SOAP. Web-сервис выполняет запрошенную клиентом функцию и отправляет результат клиенту, также применяя SOAP.

Стоит отметить, что каталог не является обязательным компонентом применения Web-сервисов: если клиенту известно описание сервиса и его адрес, он может обращаться к сервису без взаимодействия с каталогом; также и публикация в каталоге не является обязательной для сервиса, если он дает знать о себе своим клиентам каким-то иным способом. [4]

При использовании Web-сервиса происходит передача данных его методов по сети с помощью протоколов. Для того чтобы клиентское приложение могло правильно обмениваться информацией с методами Web-сервиса, оно должно выполнить следующие четыре действия:

* Проверить существование Web-сервиса. Поставщиков Web-сервиса с определенными функциональными возможностями можно найти в каталоге, таком как каталог сервисов UDDI. В этом каталоге имеются URL-адреса веб-сайтов поставщиков служб.
* Обнаружить Web-сервиса. При наличии URL-адреса поставщика выполняется обнаружение Web-сервиса, чтобы получить конкретные сведения о каждой Web-сервисе, доступной по этому URL-адресу. Информация о каждом Web-сервисе возвращается клиенту в виде описания службы — документа XML, в котором она описана на языке WSDL. Описание Web-сервиса четко определяет способ взаимодействия с ней.
* Получив описание Web-сервиса, создать прокси-класс, способный взаимодействовать с ее методами на основании точного определения в ее описании.
* Создать клиентское приложение, вызывающее методы прокси-класса. Методы прокси-класса могут взаимодействовать с методами Web-сервиса через Интернет с помощью стандартных для отрасли протоколов.

Обнаружив существование Web-сервиса с помощью обнаружения Web-сервисов, можно просмотреть информацию об этой службе и реализуемых в ней методах в формате, более наглядном, чем описание службы. [5]

Web-сервисы могут использоваться множеством различных клиентских приложений. Взаимодействовать с Web-сервисом может любое веб-приложение, даже другой Web-сервис. Клиент Web-сервиса — это не обязательно клиентское приложение: на практике многие клиенты являются серверными приложениями, такими как веб-формы и другие Web-сервисы.

СПИСОК ЛИТЕРАТУРЫ

[1] Шилдт, Г. C# 4.0: Полное руководство / Г. Шилдт. – М. : ООО Издательский дом Вильямс, 2011. – 1056 с.

[2] Троелсен ,Э. Язык программирования C# 2010 и платформа .NET 4 5-е издание / Э. Троелсен – М.: ООО Издательский дом Вильямс, 2011. – 1392с

[3] Нейгель, К. C# 4.0 и платформа .NET4 для профессионалов / К. Нейгель [и др.]: ООО Издательский дом Вильямс, 2011. – 1440с.

[4] Шорт С. Разработка XML Web-сервисов средствами Microsoft.NET / С. Шорт [и др.]: OOO БХВ-Петербург, 2003г – 480с.

[5] Библиотека MSDN [Электронный ресурс]. – Электронные данные. – Режим доступа: http://msdn.microsoft.com/.