UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

DESARROLLO DE PRACTICAS EN LA TECNOLOGÍA .NET PARA EL CURSO DE "CICLO DE DESARROLLO DE SOFTWARE"

REPORTE DE ACTIVIDAD DOCENTE

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

FRANCISCO DOMINGUEZ MENDIETA

TUTORA:

HANNA JADWIGA OKTABA

2010

Alumno

Domínguez
Mendieta
Francisco
56 76 86 53
Universidad Nacional Autónoma de México
Facultad de Ciencias
Ciencias de la Computación
097173577

Tutor

Dra.
Hanna Jadwiga
Oktaba

Sinodal 1

M. en C.
María Guadalupe Elena
Ibargüengoitia
Gonzales

Sinodal 2

Dra.
Amparo
López
Gaona

Sinodal 3

Dra.
Hanna Jadwiga
Oktaba

Sinodal 4

M. en C.
Cecilia
Pérez
Colín

Sinodal 5

Mtra.
Ana Luisa
Solís
González-Cosio

Datos del trabajo escrito

DESARROLLO DE PRACTICAS EN LA TECNOLOGÍA .NET PARA EL CURSO DE "CICLO DE DESARROLLO DE SOFTWARE"
126 p.
2011

RESUMEN

Este proyecto tiene como objetivo la creación de las prácticas de laboratorio necesarias para realizar las actividades del curso *Ciclo de Desarrollo de Software*. La finalidad de estas prácticas es guiar al alumno a través de este curso utilizando herramientas de software de Microsoft, cumpliendo así, con el convenio efectuado entre la UNAM y la empresa Microsoft México en el mes de Marzo de 2007. De acuerdo al temario del curso el alumno deberá realizar el análisis, diseño e implementación de un sistema de software siguiendo una metodología adecuada. Para cumplir este propósito las prácticas desarrolladas en este proyecto cubren todos los temas necesarios, desde la especificación de requerimientos, análisis y diseño del sistema, hasta la implementación del mismo. La plataforma principal que se utiliza en el transcurso de estas prácticas es el Framework .Net de Microsoft y las herramientas de software que se emplearon son Visio y Visual Studio, ambas de la misma empresa. Estas prácticas le darán al alumno la oportunidad de iniciarse en un nuevo ambiente de desarrollo, incrementando sus conocimientos y proporcionándole una mayor amplitud en el campo laboral.

ÍNDICE

INTRODUCCIÓN

Cada cierto tiempo, los desarrolladores de software de hoy en día, tienen que realizar una autoevaluación de sus conocimientos con el fin de mantenerse actualizados en las nuevas tecnologías. Los lenguajes y las arquitecturas que antes eran utilizadas como balas de plata del desarrollo de software, eventualmente son reemplazados por algo mejor o al menos por algo más nuevo. A pesar de la frustración que se pueda sentir cuando se renueva el conocimiento, este proceso es inevitable.

Debido a la gran diversidad de software, lenguajes de programación y entornos de trabajo que existen en el área del desarrollo de software, es de suma importancia que un alumno de la Facultad de Ciencias tenga la capacidad de desarrollar sistemas de software en distintos ambientes y plataformas.

Bajo esta premisa la Universidad Nacional Autónoma de México a través de la Facultad de Ciencias y la empresa Microsoft México realizaron un convenio de colaboración en el mes de marzo del año 2007 (convenio 19403-1688-29-XI-06 Microsoft-Facultad de Ciencias, UNAM), este convenio tuvo como objetivo apoyar y facilitar la investigación y docencia académica sobre el uso y la aplicación de nuevas tecnologías.

Entre las cláusulas de este convenio se señala que:

- La UNAM a través de la Facultad de Ciencias, se compromete a la creación y desarrollo del contenido curricular requerido para impartir las materias de:
    a. Aplicaciones de Internet Web 2.0
    b. Videojuegos
    c. Ciclo de Desarrollo de Software
- La estructura didáctica del material creado, estará desarrollada bajo el concepto de materiales de aprendizaje reutilizables.
- Los productos y tecnologías que servirán de base para la realización de los materiales, serán sobre productos y tecnologías Microsoft, tales como: Microsoft .NET Framework 2.0, Visual Studio 2005, y Microsoft SQL Server 2005.

El objetivo de este proyecto, es cumplir con la cláusula del convenio que se refiere a la elaboración de material educativo. Específicamente, la creación y desarrollo de las prácticas de laboratorio necesarias para impartir el curso "**Ciclo de desarrollo de Software**". Estas prácticas guiarán al alumno a través del proceso de desarrollo de software utilizando técnicas de ingeniería de software mediante el uso de tecnologías Microsoft.

A continuación se presenta una visión general de lo que es el *Ciclo de desarrollo de software*, así como de las metodologías que se utilizan en este curso, para dar paso a la explicación de cómo se puede hacer uso de los productos y tecnologías de Microsoft para llevar a cabo este proceso, específicamente el uso del entorno de trabajo (*Framework*) Microsoft .NET 2.0.

CAPÍTULO I. Ciclo de desarrollo de Software

En la actualidad, los sistemas de software se han convertido en un elemento indispensable en muchas de las actividades del ser humano, provocando la necesidad de perfeccionar el proceso por medio del cual el software es creado. Por este motivo es de vital importancia conocer cuáles son los ciclos y fases en los que se divide el periodo de vida del software.

1.1 Introducción al ciclo de vida del software

La creación de productos de software tiene un ciclo de vida que de forma muy general se puede dividir en dos etapas:

Etapa de concepción y desarrollo

- Se definen las necesidades que deberá cubrir el software.
- Se analiza y diseña el producto que las cumplirá
- Se construye y prueba el producto

Etapa de operación, mantenimiento y retiro

- Se pone en operación
- Se va modificando y mejorando
- Eventualmente se deshecha

En este capítulo sentaremos las bases de la ingeniería de software y nos enfocaremos en la etapa de concepción y desarrollo del software.

1.2 El proceso de desarrollo del software

El *Proceso de desarrollo del software* es el conjunto de actividades que se necesitan para transformar las necesidades de una persona o empresa en un producto de software. Permite que los desarrolladores sepan qué hacer, cuándo, cómo y quién es el responsable.

Un **Proceso de desarrollo del software** está compuesto por fases y estas a su vez están compuestas de al menos una actividad que tiene asociado uno o varios productos y uno o varios roles. Un rol es responsable de al menos una actividad.

- Las **Fases** constituyen pasos significativos del proceso de desarrollo del software. Tienen un objetivo dentro del desarrollo. Para cada fase se identifican: los roles, actividades y productos que son necesarios para cumplir el objetivo de la fase.
- **Actividades** definen las acciones que se llevan a cabo en el desarrollo del software y utilizan y/o generan los productos.
- **Productos** son las entradas y salidas de las actividades. Pueden ser documentos, diagramas de diseño, código, planes de pruebas, reportes, manuales de usuario, o conjuntos de ellos.
- **Roles** son los responsables por llevar a cabo las actividades del proceso.

El proceso de desarrollo del software en este curso está basado en el Proceso Unificado, el cual es un marco de desarrollo de software que se caracteriza por estar **guiado por los casos de uso**, **centrado en la arquitectura** y por ser **iterativo e incremental.**

- **Guiado por los casos de uso**:
  Un caso de uso es una funcionalidad del sistema que proporciona al usuario un valor o servicio. Un usuario es una persona o un sistema que interactúa con el software. Por lo que los casos de uso guían el desarrollo del software para que cumpla con las necesidades del usuario.

- **Centrado en la arquitectura**
  El Proceso Unificado asume que no existe un modelo único que cubre todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema. La analogía con la construcción es clara, cuando construyes un edificio existen diversos planos que incluyen los distintos servicios del mismo: electricidad, fontanería, etc.

- **Iterativo e incremental**
  Una iteración es la ejecución de todos los pasos del ciclo de desarrollo. Un Incremento es la evolución que va teniendo el producto a lo largo del tiempo. En el desarrollo se escogen algunos casos de uso iníciales para una iteración y en versiones posteriores del software se incrementa incorporando otros casos de uso. Las iteraciones se deben planear y controlar.

Una iteración del ciclo de desarrollo consta de las siguientes fases:

- **Especificación de requerimientos**. El objetivo de esta fase es entender, capturar y especificar los requerimientos para tener una descripción clara y no ambigua de lo que será el producto. Esta especificación debe proporcionar criterios para validar el producto terminado. Se utilizan los casos de uso para especificar los requerimientos.

- **Análisis**. Se analizan los requisitos para tener un mejor entendimiento de lo que se pretende crear y se construye el modelo del análisis para identificar los elementos que servirán de base para estructurar todo el sistema. Se establecen las clases con que se construirá el software. Se construyen dos modelos: la vista estática (diagramas de clases) y dinámica (diagramas de secuencia y de estados).

- **Diseño**. Los objetivos de esta fase son: describir las partes de las cuales se va a componer el sistema y mostrar sus relaciones en la arquitectura, se identifican los paquetes principales del software y la forma en que se distribuirán en las computadoras que intervienen en una aplicación.

- **Construcción**. El objetivo de esta fase es hacer la construcción del software y entregar el código probado de las unidades.

- **Integración y pruebas**. El objetivo de esta fase es hacer la integración del sistema y probar que cumpla con sus requerimientos.

CAPÍTULO II. Metodologías y lenguajes del curso

Ya que conocemos las fases en las que se divide el ciclo de desarrollo de software, podemos profundizar en las metodologías y lenguajes que se utilizarán en este curso.

El entorno de programación implica tanto el lenguaje de programación como el empleo de una determinada metodología. Los lenguajes de programación se ven influidos en gran manera por la forma en la que los profesionales piensan que se debe programar. De esta manera se crea un conjunto de reglas para simplificar la tarea de programación.

Estas reglas son modelos que proporcionan técnicas, que a su vez deben aplicarse en el diseño e implementación de los programas. Estas técnicas nos indican la forma de resolver los distintos problemas que surgen a la hora de programar. A estos principios se les denomina metodologías de programación.

Las metodologías de programación son modelos sobre cómo diseñar e implementar los programas. Diferentes modelos tienen como resultado diferentes técnicas. **Que cada técnica sea distinta no implica que una sea la verdadera y las demás falsas.** Por el contrario, las metodologías se complementan entre sí. Lo que todas las metodologías tienen en común es la premisa de que hay que partir de abstracciones que corresponden a elementos del problema a resolver, y que la implementación de la solución se debe realizar mediante un conjunto de módulos preferiblemente reutilizables.

2.1 Paradigma Orientado a Objetos

Las metodologías orientadas a objetos se centran en las estructuras de datos. La base de esta metodología es que una estructura de datos debe contener las operaciones que las modifican. La técnica que se utiliza para obtener esta *abstracción de datos* es la encapsulación de los mismos en una estructura conocida como clase.

El acceso a los datos contenidos en la clase se realiza mediante las operaciones que la propia clase define. Por tanto, la metodología orientada a objetos complementa el punto de vista procedural de operaciones realizadas sobre un flujo de datos, al asociar a cada dato el conjunto de operaciones que lo modifican.

No es objetivo de este proyecto profundizar en el concepto del paradigma orientado a objetos, sin embargo, resulta importante mencionar algunas de las ventajas más importantes que esta metodología tiene y debido a las cuales la programación orientada a objetos ha demostrado ser una metodología que permite a los desarrolladores cumplir de forma adecuada con la mayoría de los requerimientos que presenta hoy en día el desarrollo de software, entre los cuales están:

- **Claridad:** Al ligar de forma evidente la estructura de la información con los procedimientos que la manipulan, los programas ganan en claridad a la hora de desarrollarlos y mantenerlos. Esto supone una ventaja frente a los lenguajes procedurales.

- **Complejidad:** Cuando la complejidad de un problema es abarcable por una sola persona, resolverlo con una herramienta u otra no aporta grandes ventajas. Pero cuando este desarrollo la tiene que realizar un equipo grande, debe existir una forma para aislar partes de problema. Uno de los problemas más comunes, y a su vez más simples de solucionar en el diseño de grandes sistemas, es el nombre que se da a las funciones y qué tipo de datos manipulan éstas.

- **Tamaño:** Las aplicaciones orientadas a objetos son ideales para la realización de programas de gran tamaño. Las facilidades de encapsulación y asociación de las funciones a los datos que manipulan, simplifican el proceso de desarrollo.

- **Relación entre Datos:** Por el mismo motivo se verán beneficiados aquellos programas que impliquen una relación compleja entre los datos. Este tipo de complejidad permite la utilización de todas las ventajas de los lenguajes de programación orientados a objetos. Propiedades como la herencia (donde los objetos pueden heredar estructura y operaciones de objetos predecesores), la encapsulación, etc. Muestran en este tipo de programas todas sus ventajas.

- **Rapidez:** En este aspecto, los lenguajes orientados a objetos muestran una clara desventaja frente a otros lenguajes que se acercan más a las especificaciones de la máquina. Sin embargo, si la rapidez es crítica, se puede elegir un lenguaje de programación como C++ que aporta toda la funcionalidad de los lenguajes orientados a objetos con la rapidez y la compatibilidad del lenguaje C.

- **Gestión de recursos:** Las aplicaciones orientadas a objetos demandan normalmente más recursos del sistema que las aplicaciones procedurales. La creación dinámica de objetos, que ocupa un lugar en la memoria del ordenador puede acarrear graves problemas. Una de las soluciones, que incluye alguno de los lenguajes de POO (Programación Orientada a Objetos), es liberar a menudo el espacio que los objetos dejan de utilizar. Este procedimiento de optimización se conoce como *garbage collection* (recolección de basura, implementado en java), minimiza los efectos de la creación dinámica de objetos.

- **Interfaz de usuario:** La interfaz de usuario es uno de los aspectos más importantes en la programación actual. La aparición de sistemas de explotación que soportan una interface gráfico de usuario como *Windows*, *X-Windows* o *Presentation Manager* hace que la mayoría de los usuarios prefieran que sus programas corran bajo este tipo de interface. Este es uno de los puntos fuertes para la elección de un lenguaje POO. La mayoría de los

interfaces gráficos actuales han sido diseñados o rediseñados en base a la POO. Existen en el mercado librerías de clases que soportan todos los dispositivos de control de ventanas como menús, combo box, listas, barras de herramientas, etc.

Debido a estas ventajas **la programación orientada a objetos es la metodología de programación que se utiliza en el curso *Ciclo de Desarrollo de Software*.** En las prácticas que se desarrollaron como parte de este proyecto se utilizará como lenguaje de programación principal el Lenguaje desarrollado por Microsoft llamado *C#*.

2.2 UML

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es el estándar de la OMG (Object Management Group) para el modelado orientado a objetos y **es la herramienta de modelado que se utiliza en el curso del *Ciclo de Desarrollo de Software*** por las siguientes razones:

- Provee de un lenguaje de modelado expresivo y visual.
- Es independiente de lenguajes de programación y los procesos de desarrollo.
- Cubre los requerimientos de modelado de los sistemas actuales, por ejemplo, sistemas concurrentes y distribuidos.
- Está enfocado a proporcionar un lenguaje de modelado estándar y no a un proceso estándar.

Hay dos tipos de diagramas en UML.

- **Diagramas estructurales**: Muestran la estructura estática y los elementos del sistema. Se dividen en diagramas de: clases, objetos, componentes, paquetes y distribución.

- **Diagramas de comportamiento:** Muestran la dinámica de la funcionalidad del sistema. Se dividen en diagramas de: casos de uso, interacción, secuencia, comunicación, tiempo, actividades y estados.

Para construir buenos diagramas se tienen las siguientes recomendaciones:

- Cada diagrama debe contener un aspecto del sistema.
- Contiene lo esencial para entender ese aspecto.
- Los diagramas se mantienen consistentes con su nivel de abstracción.
- Muestran lo necesario para transmitir su semántica.

Durante la fase de *análisis* y *diseño* del ciclo de desarrollo de software se construyen diversos diagramas de UML. En las prácticas que se desarrollaron como parte de este proyecto se muestra como crear estos diagramas mediante la herramienta de Microsoft *Visio*.

CAPÍTULO III. El Framework .NET

El Framework .Net surge como la respuesta de Microsoft a la creciente competencia en el ámbito de entornos de desarrollo de software. Es un componente integral de Windows que admite la compilación y la ejecución de aplicaciones Desktop, Web y servicios Web XML. El diseño del Framework .NET está enfocado a cumplir los siguientes objetivos:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se puede almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.
- Ofrecer un entorno de ejecución de código que promueva la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.
- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en Web.
- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET se puede integrar con otros tipos de código.

El Framework .NET contiene dos componentes principales: *Common Language Runtime* y la *Biblioteca de Clases* de .NET.

El *Common Language Runtime* (CLR) es el núcleo del Framework .NET. Un motor en tiempo de ejecución que se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la comunicación remota, al mismo tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que promueven su seguridad y solidez. De hecho, el concepto de administración de código es un principio básico del motor en tiempo de ejecución. El código destinado al CLR se denomina código administrado a diferencia del resto del código que se conoce como código no administrado. El CLR es similar a la maquina virtual de Java.

La biblioteca de clases, el otro componente principal del Framework .NET, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos, hasta las aplicaciones basadas en los elementos de aplicaciones Web proporcionadas por ASP.NET, como los formularios Web Forms y los servicios Web XML.

El Framework .NET puede hospedarse en componentes no administrados que cargan el Common Language Runtime en sus procesos e inician la ejecución de código administrado, con lo que se crea un entorno de software en el que se pueden utilizar características administradas y no administradas.

ASP.NET hospeda el motor en tiempo de ejecución para proporcionar un entorno de servidor escalable para el código administrado. ASP.NET trabaja directamente con el motor en tiempo de ejecución para habilitar aplicaciones de ASP.NET y servicios Web XML.

Internet Explorer es un ejemplo de aplicación no administrada que hospeda el motor en tiempo de ejecución (en forma de una extensión de tipo MIME). Al usar Internet Explorer para hospedar el motor en tiempo de ejecución, se pueden incrustar componentes administrados o controles de Windows Forms en documentos HTML. Al hospedar el motor en tiempo de ejecución se hace posible el uso de código móvil administrado (similar a los controles de Microsoft ActiveX), pero con mejoras significativas que sólo el código administrado puede ofrecer, como la ejecución con confianza parcial y el almacenamiento aislado de archivos.

3.1 Características del CLR

El Common Language Runtime administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Estas características son intrínsecas del código administrado que se ejecuta en el Common Language Runtime.

Con respecto a la seguridad, los componentes administrados reciben grados de confianza diferentes en función de una serie de factores entre los que se incluye su origen (como Internet, red empresarial o equipo local). Esto significa que un componente administrado puede ser capaz o no, de realizar operaciones de acceso a archivos, operaciones de acceso al registro y otras funciones delicadas, incluso si se está utilizando en la misma aplicación activa.

El motor en tiempo de ejecución impone seguridad en el acceso al código. Por ejemplo, los usuarios pueden confiar en que un archivo ejecutable incrustado en una página Web puede reproducir una animación en la pantalla o entonar una canción, pero no puede tener acceso a sus datos personales, sistema de archivos o red. Además, el motor en tiempo de ejecución impone la solidez del código mediante la implementación de una infraestructura estricta de comprobación de tipos y código denominada CTS (Common Type System, Sistema de tipos comunes). El CTS garantiza que todo el código administrado es autodescriptivo. Los diversos compiladores del lenguaje de Microsoft y de otros fabricantes generan código administrado que es atendido por el CTS. Esto significa que el código administrado puede consumir otros tipos e instancias administrados, al mismo tiempo que se exige fidelidad de tipos y seguridad de tipos estrictamente.

E *CLR* controla automáticamente la disposición de los objetos, administra las referencias a éstos y los libera cuando ya no se utilizan. Esta administración automática de la memoria soluciona los dos errores más comunes de las aplicaciones: la pérdida de memoria y las referencias no válidas a la memoria. Además, el motor en tiempo de ejecución aumenta la productividad del programador. Por ejemplo, los desarrolladores pueden crear aplicaciones en el lenguaje .Net que prefieran y seguir sacando todo el provecho del motor en tiempo de ejecución, la *biblioteca de clases* y los componentes escritos en otros lenguajes por otros desarrolladores, lo que facilita enormemente el proceso de migración de las aplicaciones existentes.

El CLR es compatible con el software actual y el software antiguo. La interoperabilidad entre el código administrado y no administrado permite que los desarrolladores continúen utilizando los componentes COM y las DLL que necesiten. En el motor en tiempo de ejecución

proporciona muchos servicios estándar de motor en tiempo de ejecución, sin embargo, el código administrado nunca se interpreta, a esta característica se le llama compilación JIT (Just-In-Time) permite ejecutar todo el código administrado en el lenguaje máquina nativo del sistema en el que se ejecuta.

Por último, el motor en tiempo de ejecución se puede hospedar en aplicaciones de servidor de gran rendimiento, como Microsoft SQL Server e Internet Information Services (IIS). Esta infraestructura permite utilizar código administrado para escribir lógica empresarial, así mismo se puede hacer uso de servidores empresariales del sector que puedan hospedar el motor en tiempo de ejecución.

3.2 Biblioteca de clases de .NET Framework

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con Common Language Runtime. Esta biblioteca de clases está orientada a objetos, lo que proporciona tipos de los que su propio código administrado puede derivar funciones. Esto ocasiona que los tipos de .NET Framework sean sencillos de utilizar y reduce el tiempo asociado con el aprendizaje de sus nuevas características. Además, los componentes de terceros se pueden integrar sin dificultades con las clases de .NET Framework.

Por ejemplo, las clases de .NET Framework implementan un conjunto de interfaces que se pueden utilizar para desarrollar tus propias clases. Éstas se combinarán fácilmente con las clases de .NET Framework.

Como en cualquier biblioteca de clases orientada a objetos, los tipos de .NET Framework permiten realizar diversas tareas de programación comunes, como son la administración de cadenas, recolección de datos, conectividad de bases de datos y acceso a archivos. Además de estas tareas habituales, la biblioteca de clases incluye tipos adecuados para diversos escenarios de desarrollo especializados. Por ejemplo, se puede utilizar .NET Framework para desarrollar los siguientes tipos de aplicaciones y servicios:

- Aplicaciones de consola.
- Aplicaciones GUI de Windows (Windows Forms)
- Aplicaciones de Windows Presentation Foundation (WPF)
- Aplicaciones de ASP.NET
- Servicios Web.
- Servicios de Windows
- Aplicaciones orientadas a servicios utilizando Windows Communication Foundation (WCF).
- Aplicaciones habilitadas para el flujo de trabajo utilizando Windows Workflow Foundation (WF).

Por ejemplo, las clases de Windows Forms son un conjunto completo de tipos reutilizables que simplifican enormemente el desarrollo de interfaces GUI para Windows. Si se escribe una aplicación Web Form de ASP.NET, se utilizaran las clases de formularios Web Forms.

CAPÍTULO IV. Herramientas para el desarrollo de Software

Microsoft provee un conjunto de herramientas que podemos utilizar para llevar a cabo las actividades que se realizan en el Ciclo de Desarrollo de Software, a continuación se mencionan las herramientas que se utilizarán para desarrollar las prácticas del curso Ciclo de Desarrollo de Software.

4.1 Microsoft Visual Studio

Al igual que Netbeans y Eclipse, Visual Studio es un IDE (Integrated Development Environment) con la particularidad de que fue creado específicamente para el Framework .NET. El entorno de desarrollo que provee Visual Studio permite a los desarrolladores de .NET crear aplicaciones Windows, sitios y aplicaciones Web, así como servicios Web en cualquier plataforma .NET, así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles. Los ejercicios de las prácticas realizadas en este proyecto fueron desarrollados en Visual Studio.

4.2 Microsoft Internet Information Services (IIS)

Es un software que gestiona un conjunto de servicios, por medio de los cuales se puede crear un Servidor Web, el IIS maneja los protocolos FTP, SMTP, NNTP y HTTP/HTTPS. Entre los sistemas de su tipo, IIS se ha convertido en uno de los más extendidos, debido a su fuerte penetración en los servidores que funcionan sobre la plataforma Windows, haciendo fuerte competencia a los servidores basados en plataformas UNIX.

IIS admite la creación, configuración y administración de sitios Web desarrollados en distintas plataformas, por ejemplo para los lenguajes de Microsoft ASP (Active Server Pages) y ASP.NET, también pueden ser incluidos los de otros fabricantes, como PHP y Perl.

4.3 Microsoft SQL Server

Es un sistema manejador de bases de datos, similar a MySQL, Oracle, PostgreSQL, etc. Fue desarrollado por Microsoft y está basado en el modelo relacional. Sus lenguajes para realizar consultas son el Transact-SQL y ANSI SQL.

Entre sus principales características están:

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Soporta procedimientos almacenados.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información.
- Permite administrar información de otros servidores de datos.
- Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.

4.4 Microsoft Visio

Es una herramienta para realizar diagramas avanzados de distintos tipos, por ejemplo diagramas de bases de datos, diagramas de oficinas, diagramas de flujo, y más.

Como se ha mencionado anteriormente, en el Ciclo de Desarrollo de Software las fases del análisis y diseño son sumamente importantes. En estas etapas se realizan algunos diagramas UML, por ejemplo: el diagrama de clases, los diagramas de secuencia y el diagrama de paquetes entre otros. Visio es el programa que se ajusta a las necesidades que surgen en estas fases del ciclo de desarrollo, debido a que provee un extenso conjunto de herramientas que permiten crear diagramas UML.

CAPÍTULO V. Guía de instalación

Como se mencionó anteriormente, las herramientas de Microsoft necesarias para el Ciclo de Desarrollo de Software son: IIS, SQL Server, Visual Studio y Visio, sin embargo Visual Studio 2005 contiene una versión integrada de IIS y SQL Server en su ambiente de trabajo, que permiten a un desarrollador correr aplicaciones Web con conexión a bases de datos sin necesidad de contar con las versiones completas de estos programas instalados en su computadora. Por este motivo el IDE Visual Studio 2005 y la herramienta Visio son los únicos programas que será necesario instalar en este curso.

5.1 Instalación de Microsoft Visio

La instalación del software Microsoft Visio es sumamente sencilla, básicamente deben seguirse los siguientes pasos:

- Ejecutar el programa de instalación
- Aceptar los términos y condiciones de uso
- Presionar el botón Instalar

5.2 Instalación de Microsoft Visual Studio 2005

Una de las ventajas de trabajar con el Framework .Net es que se ha desarrollado un estándar en la mayoría de sus componentes, eliminando la necesidad de configurar los elementos del entorno de trabajo. Por esta razón la instalación de Visual Studio 2005 es prácticamente igual de simple que con cualquier otro programa.

- Ejecutar el archivo de instalación
- Dar click en Instalar Visual Studio 2005
- Seleccionar la casilla para aceptar los términos del contrato
- Seleccionar la opción Instalación predeterminada
- Dar click en instalar

CAPÍTULO VI. Prácticas del curso Ciclo de Desarrollo de Software

A continuación se presentan las prácticas realizadas en este proyecto.

6.1 Práctica 1 - Introducción a Microsoft Visio

Objetivos

- Familiarizarse con la interfaz de *Microsoft Visio*.
- Aprender que son las formas, como usarlas y como conectarlas.

Introducción

Microsoft Visio es un sistema de software de dibujo vectorial para Windows. Las herramientas que lo componen permiten realizar diagramas para oficinas, diagramas de bases de datos, diagramas de flujo de programas, diagramas de UML y otros.

Visio cuenta con una plantilla llamada *Diagrama de modelo de UML*. Esta plantilla proporciona todas las herramientas necesarias para crear modelos orientados a objetos de sistemas de software complejos. La solución incluye las herramientas, formas y funcionalidades siguientes:

- El Explorador de modelos de UML, que proporciona una vista de árbol del modelo y un medio de desplazarse entre vistas.
- Formas inteligentes predefinidas que representan los elementos de la notación de UML y admiten la creación de los tipos de diagramas de UML. Las formas se programan para comportarse de modo que sean coherentes con la semántica de UML.
- Acceso a los cuadros de diálogo propiedades de UML, donde el usuario puede agregar nombres, atributos, operaciones y otras propiedades a elementos de UML.
- La capacidad de aplicar ingeniería inversa a los proyectos creados en *Microsoft Visual Studio.NET* y generar modelos de estructura estática de **UML**.

Los diagramas de UML creados por el usuario conforman diferentes vistas del sistema modelado y se muestran en forma de árbol en el explorador de modelos.

En *Visio*, existe una forma para cada elemento de la notación UML. Estas formas están organizadas en contenedores llamados **stencils** y se comportan como se espera que lo haga el elemento de UML. Por ejemplo, la forma Clase tiene tres compartimentos para el nombre, los atributos y las operaciones. Esta forma se expande automáticamente cuando se agregan valores. Todas las formas de Visio tienen un conjunto de propiedades a las que se puede acceder al dar doble click sobre la forma.

Las formas de UML deben estar pegadas correctamente entre sí para que se comuniquen con el modelo. Cuando se conectan formas de UML, la plantilla Diagrama de modelo de UML ayuda al usuario a conectarlas correctamente mediante la presentación de un contorno rojo alrededor de los puntos de conexión o de toda la forma.

Desarrollo

- Inicia el programa *Microsoft Visio*.
- Selecciona: **Archivo -> Nuevo -> Software y base de datos -> Diagrama modelo de UML.** Tu vista debe ser similar a la de la *Figura P1.1*. La página en blanco se llama área de dibujo.

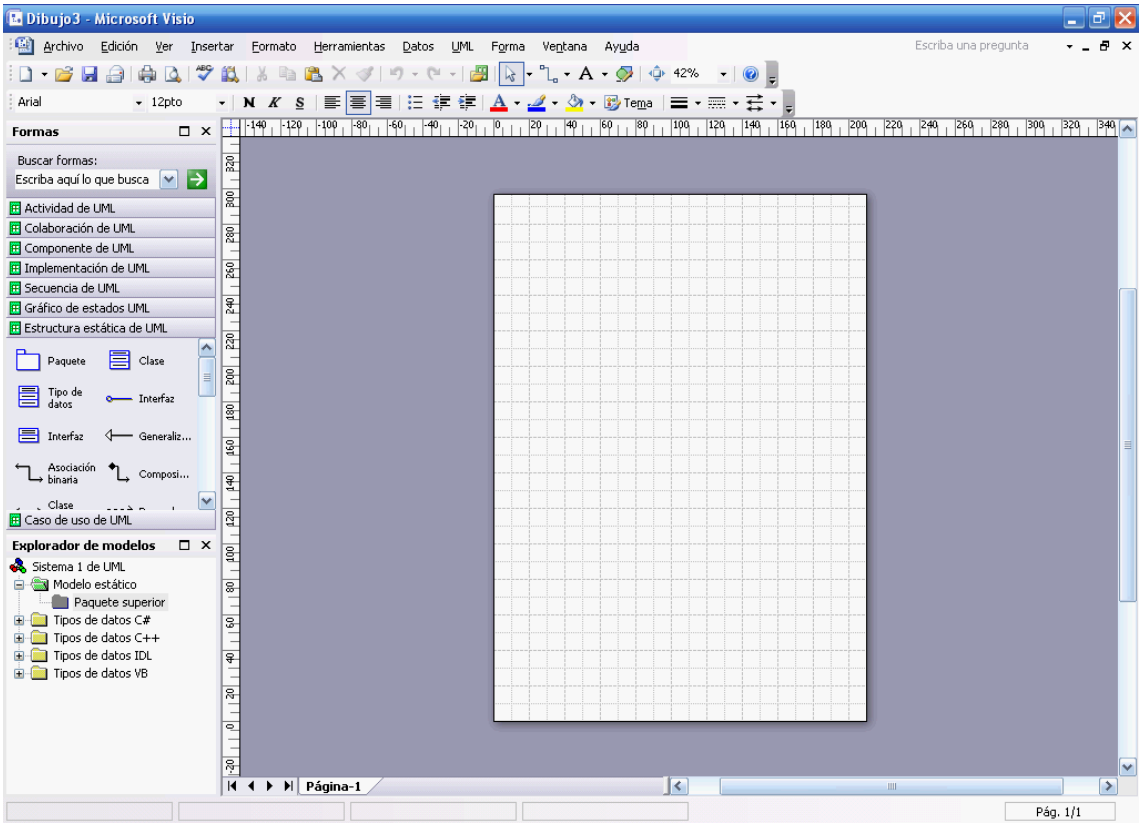


Figura P1.1

- Explora la ventana *Formas* para revisar que elementos de la notación de UML pueden ser representados.

Una forma de *Visio* puede ser tan simple como una línea o tan compleja como un calendario. Una forma de *Visio* puede ser unidimensional (1D) o bidimensional (2D). Las formas 1D se comportan como una línea, mientras que las formas 2D se comportan como un rectángulo.

Selecciona la forma *Actor* (situada en la pestaña *Caso de Uso de UML* en la ventana *Forma) y a*rrástrala hacia el área de dibujo. Agrega la forma *Caso de Uso* al área de dibujo, de la misma forma que agregaste el *Actor.*

Notarás que ambas formas aparecen en el árbol mostrado en la ventana *Explorador de modelos* (*Figura P1.2*). Una vez que las formas están en el dibujo, puedes moverlas, cambiar su tamaño o rotarlas, dependiendo del ícono que presente el cursor al colocarse sobre una parte de la forma.

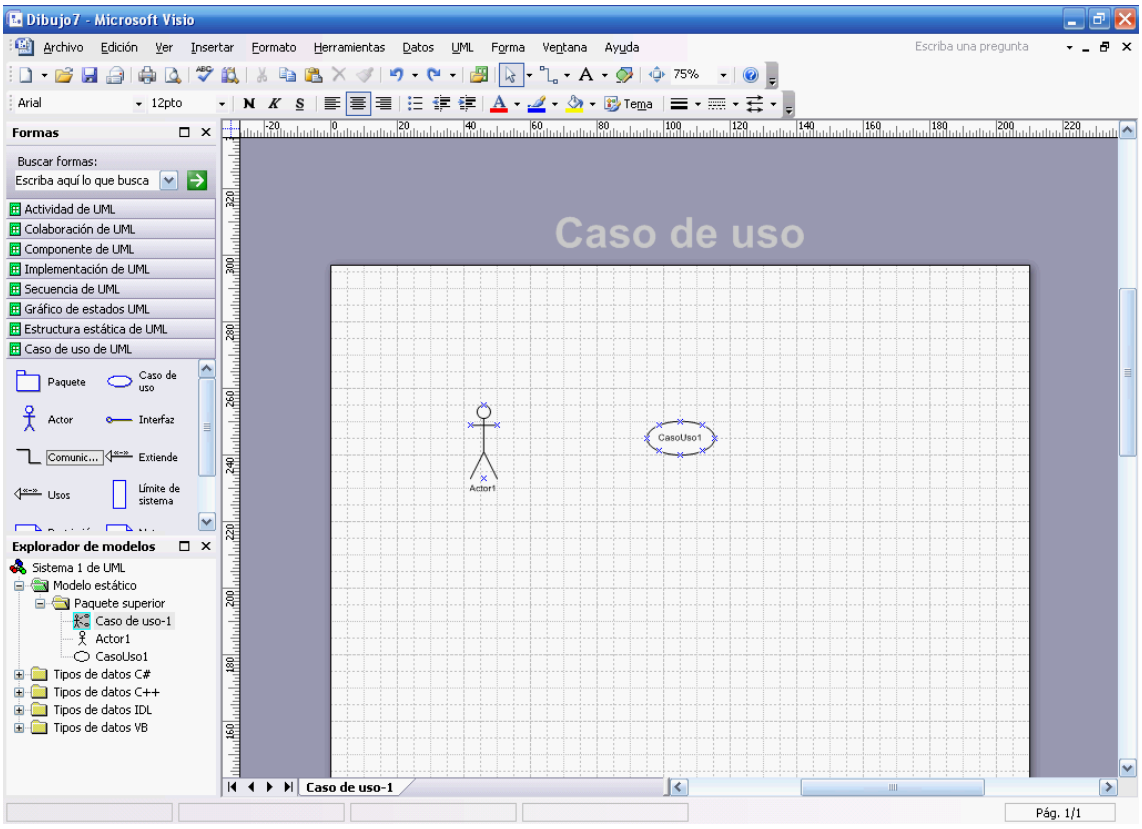


Figura P1.2

Cambia el nombre de la forma “Actor1” por “Usuario” de la siguiente manera:

- Da doble click sobre la forma “Actor1” (para acceder a sus propiedades) en el dibujo.
- Aparecerá la ventana de propiedades (*Figura P1.3*), escribe “Usuario” en el campo nombre.

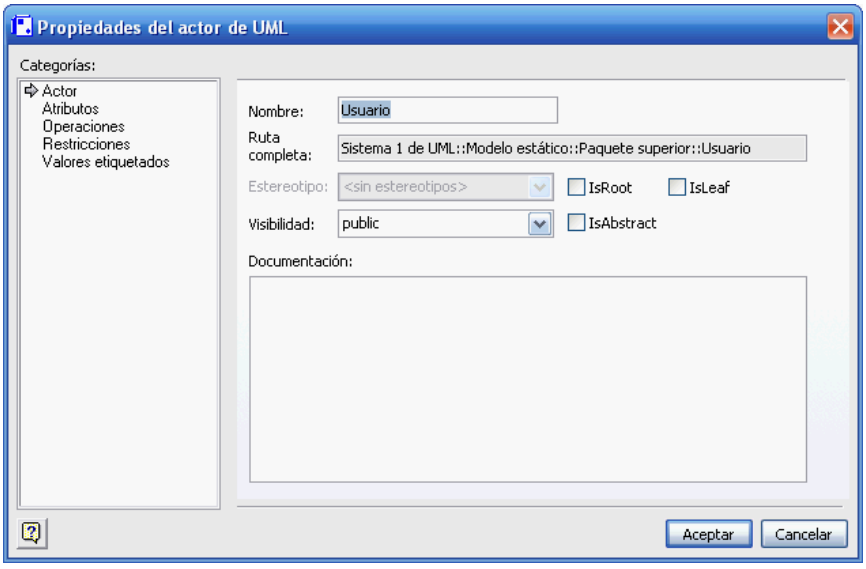


Figura P1.3

Existe otra manera de renombrar una forma:

- Selecciona la rama *Caso de Uso1* (la rama que se creó cuando agregaste la forma *caso de uso*) en la ventana *Explorador de modelos*, presiona el botón derecho del ratón y selecciona C*ambiar nombre*, cambia su nombre a “Comprar”.

Por último vas a unir estas dos formas con un conector. Para conectar formas de UML, sigue uno de los procedimientos siguientes:

- Arrastra un punto inicial o un punto final de una forma unidimensional (1D) a un punto de conexión de una forma bidimensional (2D).
- Arrastra un punto inicial o un extremo final de una forma unidimensional (1D) al centro de una forma bidimensional (2D).

Un contorno rojo indica una conexión correcta. Cuando sueltes el cursor, el extremo de la forma 1D se pondrá en color rojo para indicar que está pegado (*Figura P1.4*) y que puedes mover cualquier forma sin romper la conexión.

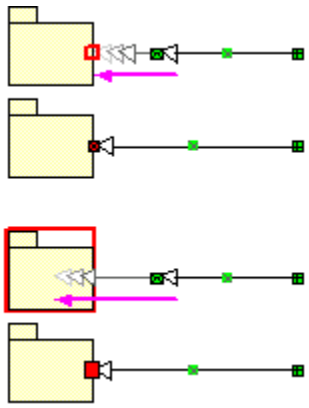


Figura P1.4

- Agrega en el dibujo una forma “Comunicados” (situada en la pestaña *Caso de Uso de UML* debajo de la forma *Actor)* de la misma forma que agregaste el *Actor* y conecta al “Usuario” con “Comprar”.
- El conector debe tener los cuadros de sus lados de color rojo (*Figura P1.5*), de lo contrario significa que las formas no están conectadas.

Debido a que la forma “Comunicados” es un conector, no aparece en la ventana *Explorador de modelos.*

- Una vez que el conector y las formas están bien “pegados”, puedes mover cualquiera de las formas y la conexión permanecerá.

Para agregar una nueva página (área de dibujo) selecciona: Insertar -> Nueva Página en el menú principal ó da click-derecho la pestaña de la página (caso de uso-1 en nuestro ejemplo) y selecciona insertar página.

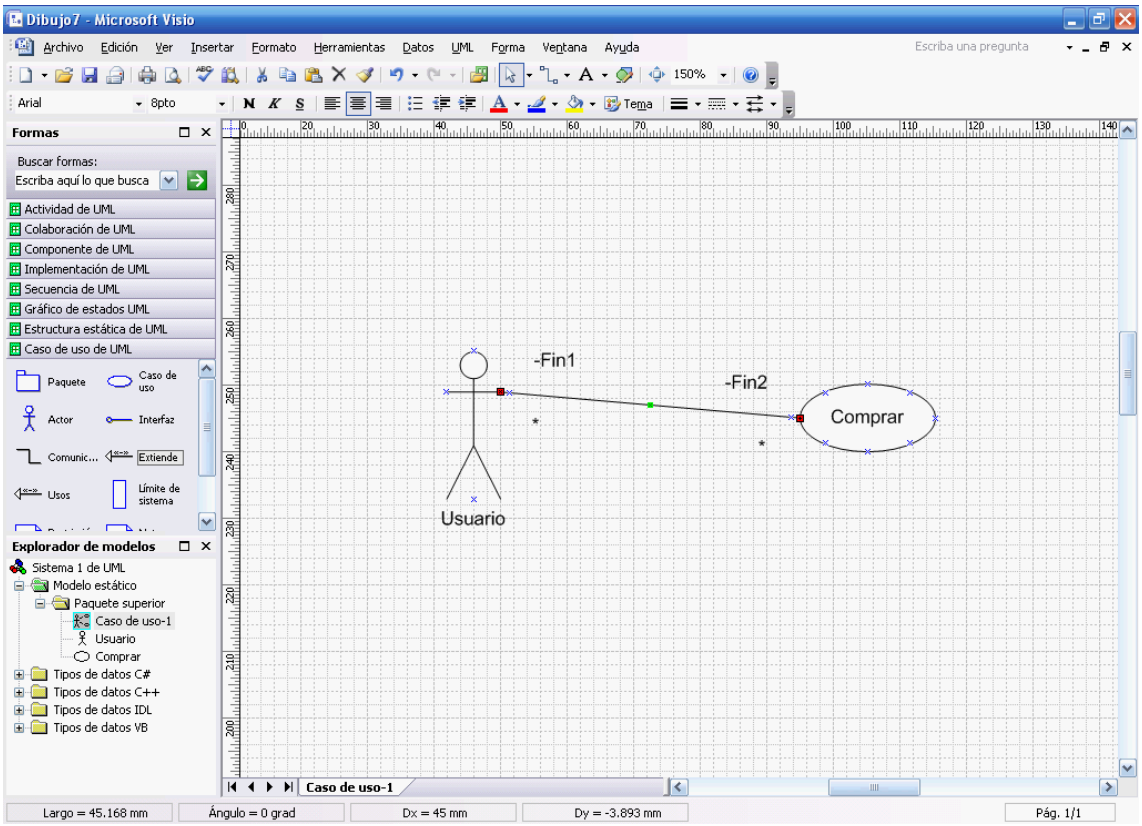


Figura P1.5

Cuestionario

- ¿Qué formas se usan en un diagrama de casos de uso?
- ¿Qué tipos de diagramas de UML puedes dibujar en *Visio*?
- ¿Te parece intuitiva la interfaz de *Microsoft Visio*?

6.2 Práctica 2 - Introducción a Microsoft Visual Studio 2005

Objetivos

- Familiarizarse con la interfaz de Visual Studio 2005.
- Realizar la primera aplicación utilizando el lenguaje de programación C#.

Introducción

En esta práctica vamos a crear una pequeña aplicación de Windows utilizando *Visual Studio 2005* y C# con el fin de familiarizarnos con la interfaz de este entorno de desarrollo.

¿Qué es Visual Studio?

Microsoft Visual Studio es un entorno integrado de desarrollo "*IDE"* (Integrated Development Environment) para sistemas Windows. El IDE es el ambiente de trabajo en el cual puedes construir aplicaciones de software. Cada herramienta que necesitarás para crear proyectos en el lenguaje C#, puede ser accedida desde *Visual Studio*. Este IDE fue desarrollado para soportar varios lenguajes de programación tales como: Visual C#, Visual C++, ASP.NET, Visual J# y Visual Basic.Net, aunque actualmente también se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma *.NET*. Se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

¿Qué es C#?

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma *.NET*.

Su sintaxis básica deriva de C y C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java, aunque incluye mejoras derivadas de otros lenguajes. C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la facilidad de programación de lenguajes como Visual Basic.

Aunque C# forma parte de la plataforma .NET, esté lenguaje de programación es independiente. Esto hace posible implementar compiladores que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX.

Desarrollo

- Ejecuta Visual Studio. La primera vez que ejecutes Visual Studio se verá muy similar a la *Figura P2.1*:

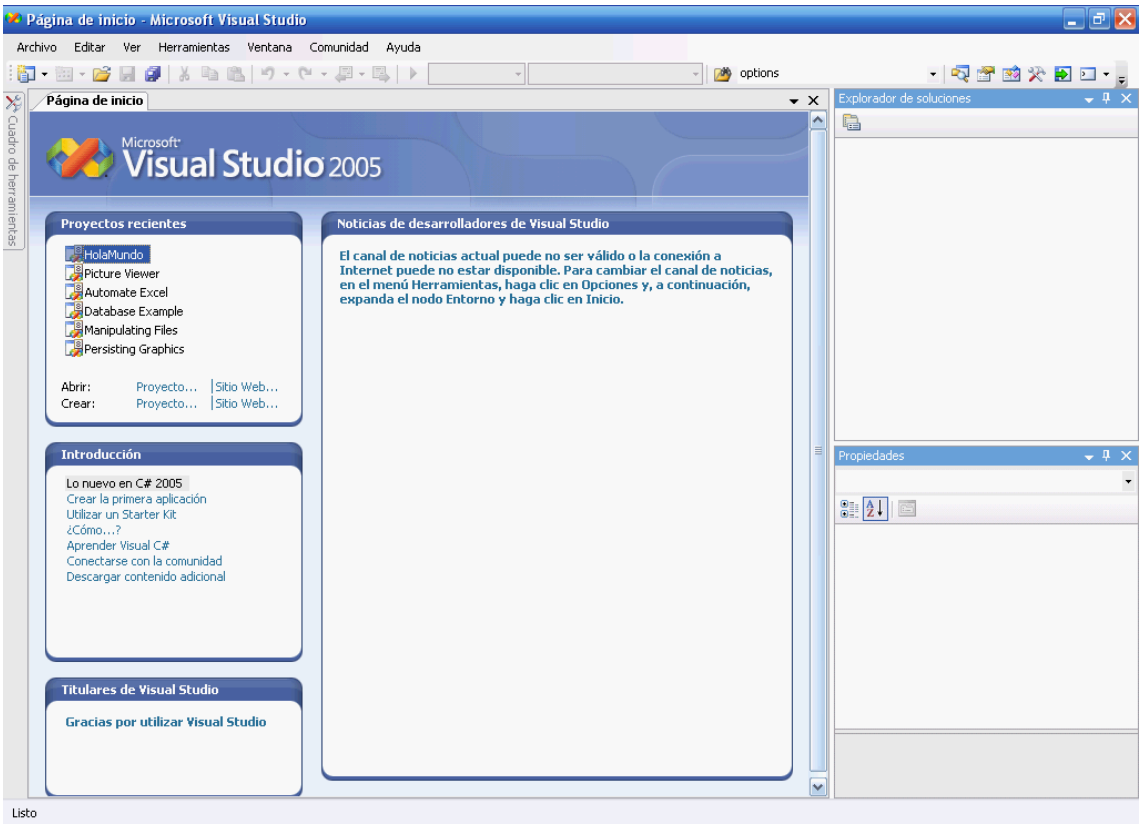


Figura P2.1 Inicio de Visual Studio 2005

- Crea un nuevo proyecto mediante alguna de las siguientes acciones:
  - Elije “Crear: Proyecto” en la ventana “Proyectos recientes” de la página de inicio.
  - Elije la opción del menú: Archivo -> Nuevo -> Proyecto.
  - Presiona: Ctrl+Mayus+N

Aparecerá el cuadro de diálogo “Nuevo Proyecto”. Aquí podrás elegir entre diferentes tipos de proyectos, tanto para *C#* como para otros lenguajes de programación soportados por *.Net Framework*.

En esta práctica vamos crear nuestra primera aplicación (Hola Mundo) utilizando *C#*, con el fin de familiarizarnos con *Visual Studio 2005*, abarcando las principales herramientas que se utilizan comúnmente para crear aplicaciones más robustas.

- Crea una aplicación de Windows, siguiendo estos pasos:
  - Selecciona “Aplicación para Windows” en el cuadro de diálogo “Nuevo Proyecto”.
  - Elije un nombre para la nueva aplicación (*Figura P2.2*).
  - Presiona Aceptar.

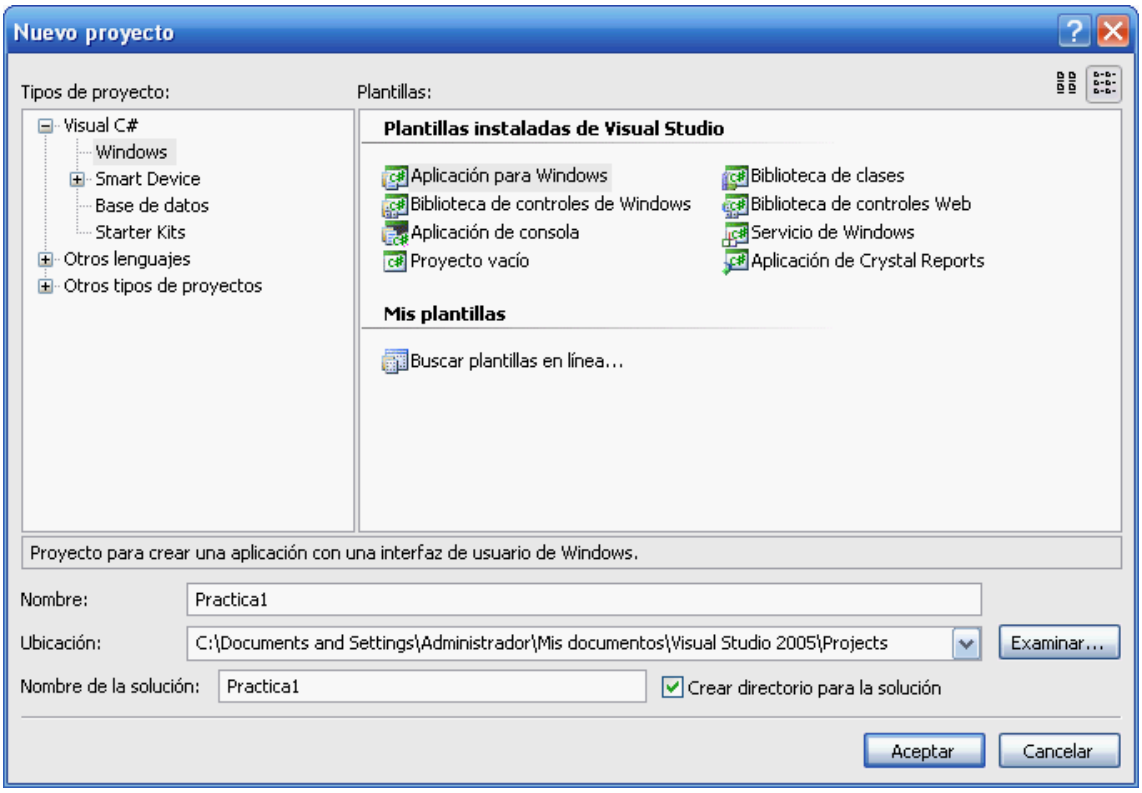


Figura P2.2

Cuando creas una nueva aplicación de Windows, *Visual Studio* crea una forma nueva (la ventana gris vacía con título form1) para que comiences a diseñar la interfaz gráfica que tendrá tu proyecto (una forma es el contenedor principal de una ventana).

La interfaz que se muestra en la *Figura P2.3*, es la que se muestra por omisión en *Visual Studio* cuando creas un nuevo proyecto de aplicación de Windows. Notarás que cuenta con un menú y una barra de herramientas (como cualquier aplicación de Windows), pero además cuenta con dos ventanas pequeñas del lado derecho. Estas dos ventanas están pensadas para brindar un mayor control sobre las aplicaciones que se van a crear.

La ventana “Explorador de soluciones” permite controlar los archivos involucrados en el proyecto de la aplicación, en nuestro ejemplo no hay muchos archivos, pero en un proyecto robusto, esta ventana es de gran utilidad para administrar y movernos entre los elementos que conforman nuestra aplicación.

La ventana “Propiedades” como su nombre lo indica, permite controlar las propiedades de cualquier objeto dentro de nuestra aplicación. Casi cualquier cosa con la que trabajamos en *Visual*

Studio es un objeto, por ejemplo los elementos que puedes colocar en una forma (cajas de texto, botones, etc.). No todos los objetos tienen apariencia física dentro de una forma, pero eso lo discutiremos en futuras prácticas.

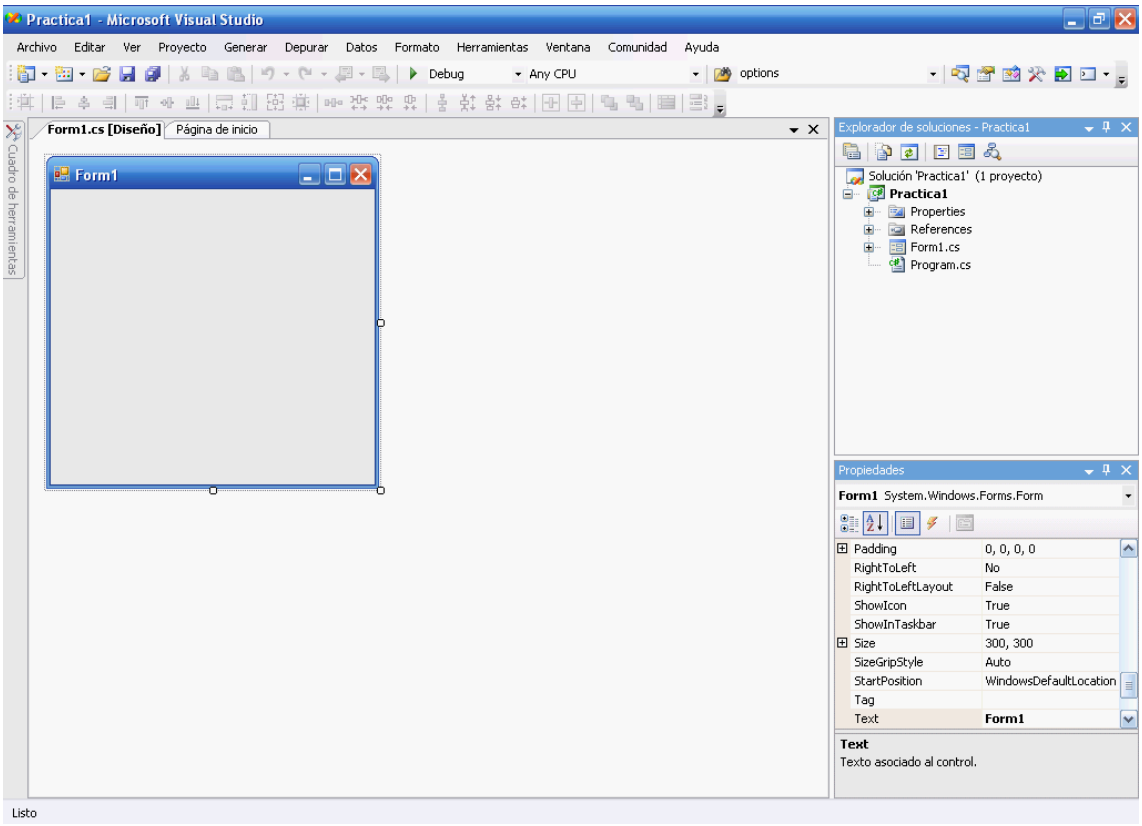


Figura P2.3

- Selecciona el archivo Form1.cs dentro de la ventana "Explorador de soluciones", *Visual Studio* le asigna este nombre por omisión a nuestra forma. Renombra el archivo a "MiForma.cs", con este nombre nos referimos a la forma dentro y fuera del código de la aplicación. Por esta razón al renombrar el archivo aparecerá una ventana como la de la *Figura P2.4*. Selecciona Sí.

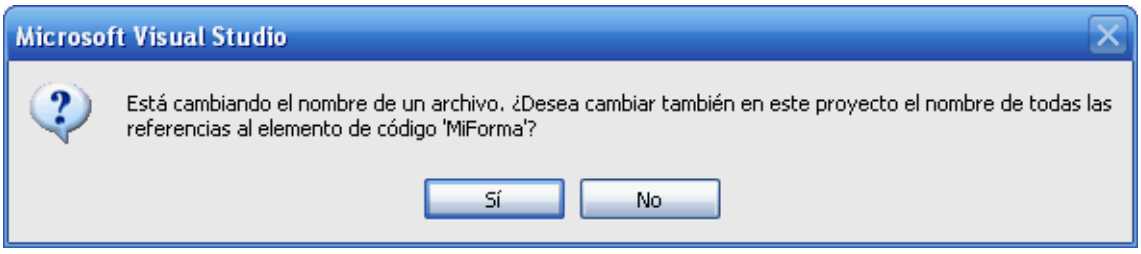


Figura P2.4

- Mientras está seleccionado el archivo “MiForma.cs”, presiona el botón “Ver código” de la ventana “Explorador de soluciones”, para ver el código que se generó al crear el proyecto (*Figura P2.5*).

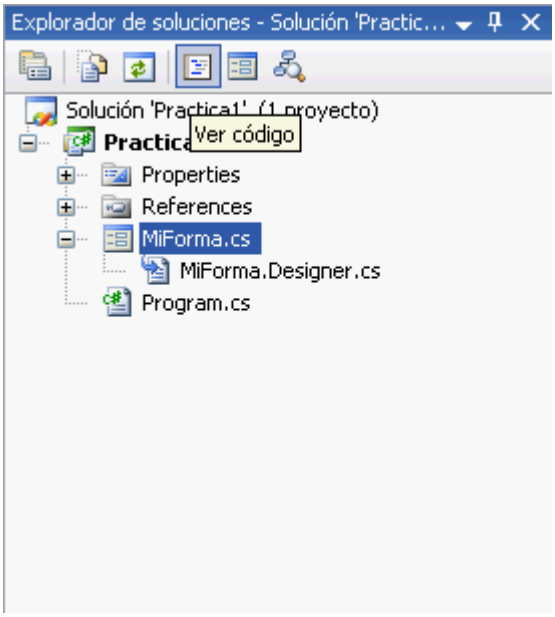


Figura P2.5

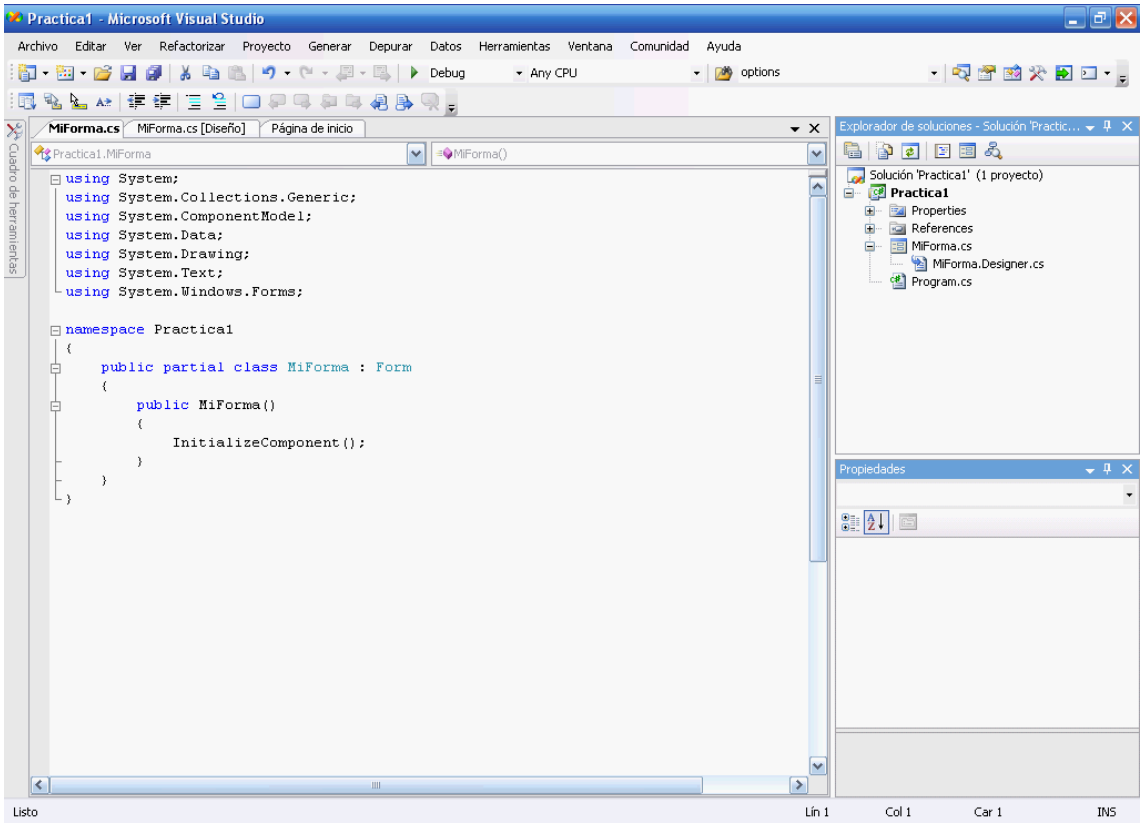


Figura P2.6

Aparecerá una nueva pestaña en la ventana principal llamada “MiForma.cs” (*Figura P2.6*). Este es el código asociado a la forma que creamos (no es objetivo de esta práctica analizar el código en *C#*).

- Regresa a la interfaz de diseño de la aplicación seleccionando la pestaña “MiForma.cs [Diseño]”.

- Da click en algún lugar del área gris en la ventana Form1 (*Figura P2.7*).

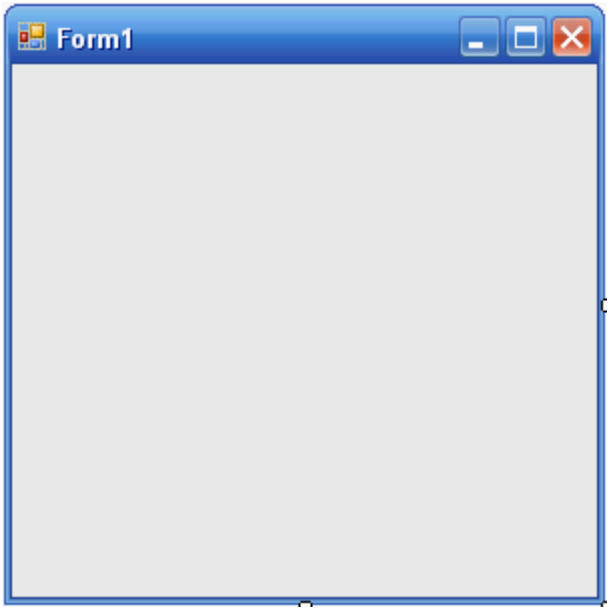


Figura P2.7

Al seleccionar un objeto, la ventana de propiedades muestra el conjunto de propiedades con las que cuenta. En la parte superior de esta ventana se muestra el nombre del objeto seleccionado (en este caso el objeto es “MiForma”), más abajo se encuentra una barra de herramientas con algunos botones.

- Da click en el botón sombreado en la *Figura P2.8* para ordenar las propiedades alfabéticamente. Busca la propiedad *Text* y escribe “Hola Mundo” (*Figura P2.8*). Notarás que la ventana de la forma (mostrada en la *Figura P2.7*), ahora tiene escrito Hola Mundo en el barra de título.

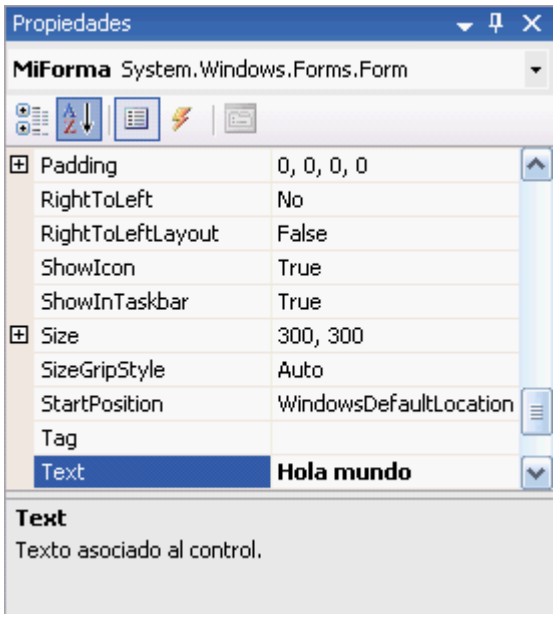


Figura P2.8

A continuación, agregaremos un botón a nuestra forma. Para hacerlo haremos uso de otra ventana de la interfaz de *Visual Studio*, el "Cuadro de herramientas" (*Figura 1.9*). El cuadro de herramientas es una ventana auto-desplegable que se encuentra del lado izquierdo. Esta ventana nos permite agregar controles de interfaz predefinidos a nuestra aplicación. Estos controles son los elementos que conforman la mayoría de las interfaces en una aplicación de Windows.

- Agrega un botón a la forma dando doble click en el control "Button" del cuadro de herramientas (*Figura P2.9*). También puedes agregar el botón de las siguientes formas:

- Da click derecho en el control "Button" y selecciona copiar. Después debes pegarlo en la forma.

- Selecciona el control "Button" y arrástralo a la forma.

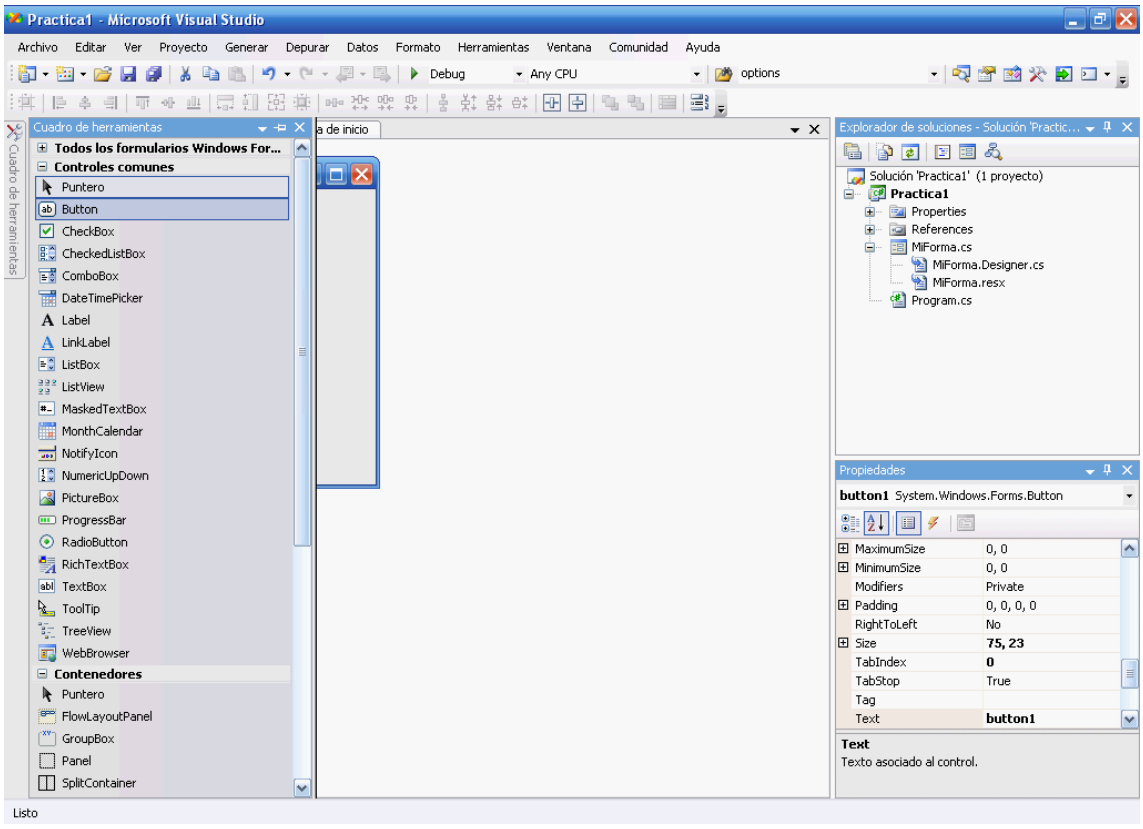


Figura P2.9

Todas las ventanas de *Visual Studio* que cuentan con una barra de titulo como la de *Figura P2.10*:

Figura P2.10

Se pueden ocultar, mostrarse como ventanas flotantes, ventanas fijas o auto-desplegables, utilizando los tres íconos del lado derecho. Con ayuda de esto puedes adaptar la interfaz de *Visual Studio* a tus necesidades.

El botón aparecerá del lado superior-izquierdo de la forma, puedes mover el botón y cambiar su tamaño, arrastrándolo con el cursor y moviendo los cuadros blancos alrededor de él. Si lo prefieres puedes poner los valores exactos en la ventana de propiedades para tener mayor precisión, como lo muestra la *Figura P2.11*

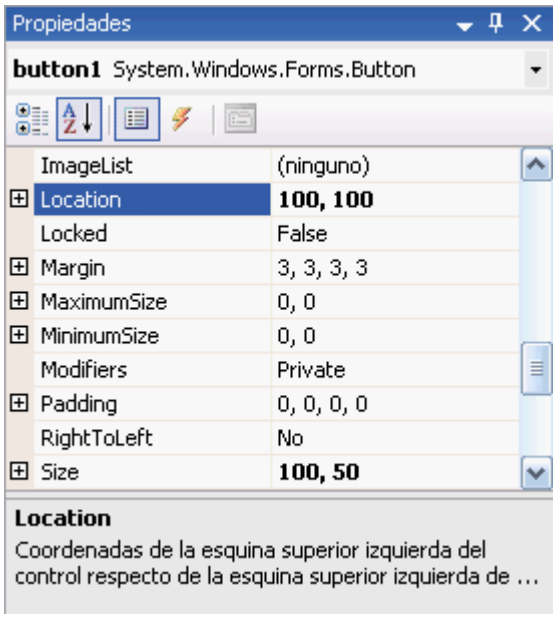


Figura P2.11

- Mueve el botón de tu aplicación para que se vea parecido al de la *Figura P2.12*. También debes sustituir la palabra "button1" por la palabra "Saludo" en la propiedad *Text* y escribir "miBoton" en la propiedad *(Name)*. Cuando quieres hacer referencia a un control dentro del código, debes hacerlo mediante su nombre, en este caso cuando nos referimos al botón que creamos será por medio del nombre "miBoton".

Figura P2.12

Da doble click en el botón que creamos (“Saludo”) dentro de la forma, se desplegará la pestaña en la que se encuentra el código (*Figura P2.13*). Observa que el código es un poco diferente al que vimos en la *Figura P2.6.* Se agregó un método llamado miBoton_Click.

Los controles de interfaz (botones, cajas de texto, etc.) funcionan mediante eventos. Un botón por ejemplo, tiene los eventos click, doble_click, mouse_over, etc. Cuando se activa uno de estos eventos, el método asociado a él, se ejecuta.

Cuando das doble click sobre un control, *Visual Studio* crea el método que se ejecutará cuando se active el evento principal del control, en nuestro ejemplo el evento principal del botón, es cuando se da click sobre él.

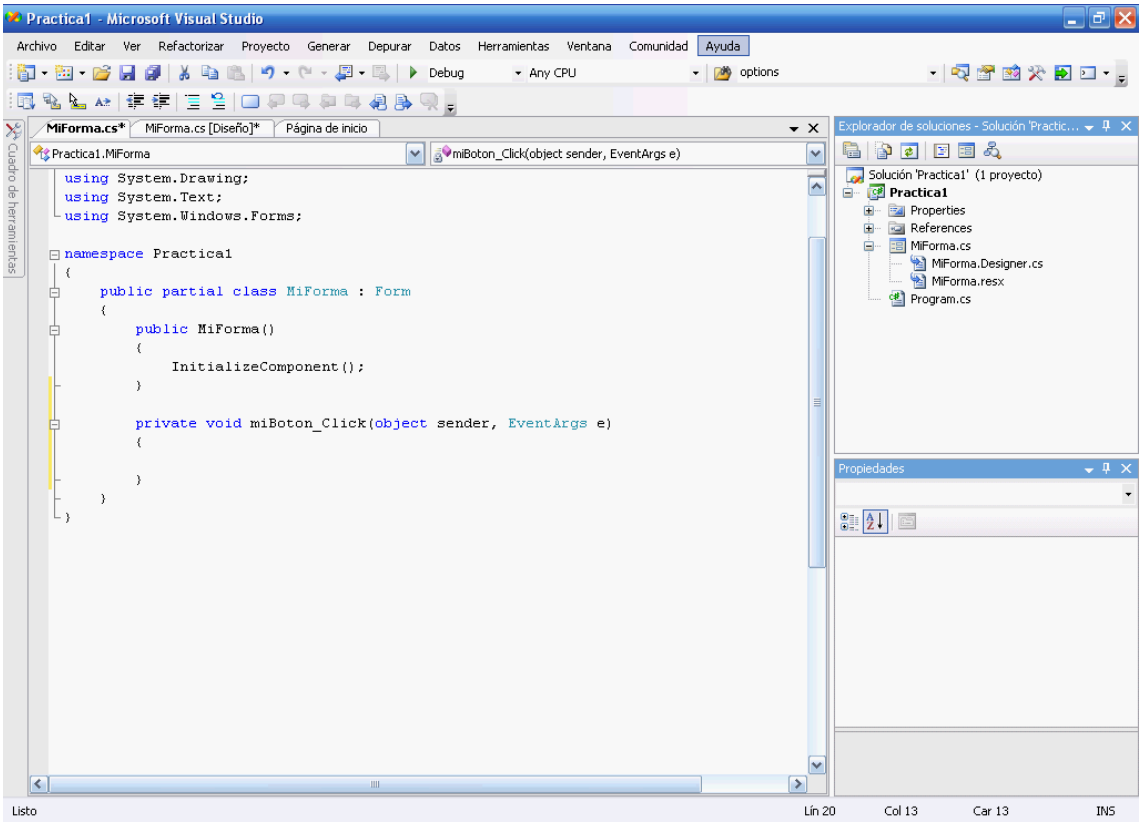


Figura P2.13

Agrega la siguiente línea de código en el método *miBoton_Click*:

```
MessageBox.Show("Hola Mundo");
```

Esta línea mostrará una ventana con el mensaje “Hola Mundo” al presionar el botón “Saludo”.

Ejecuta tu aplicación mediante alguna de estas acciones:

- Selecciona Depurar -> Iniciar depuración en el menú principal.
- Presiona el triángulo verde situado a la mitad de la barra de herramientas principal.
- Presiona la tecla F5.

Aparecerá una ventana que representa la forma que creaste en tu aplicación (*Figura P2.14*). Presiona el botón “Saludo”, debe aparecer otra ventana con la leyenda “Hola Mundo”.

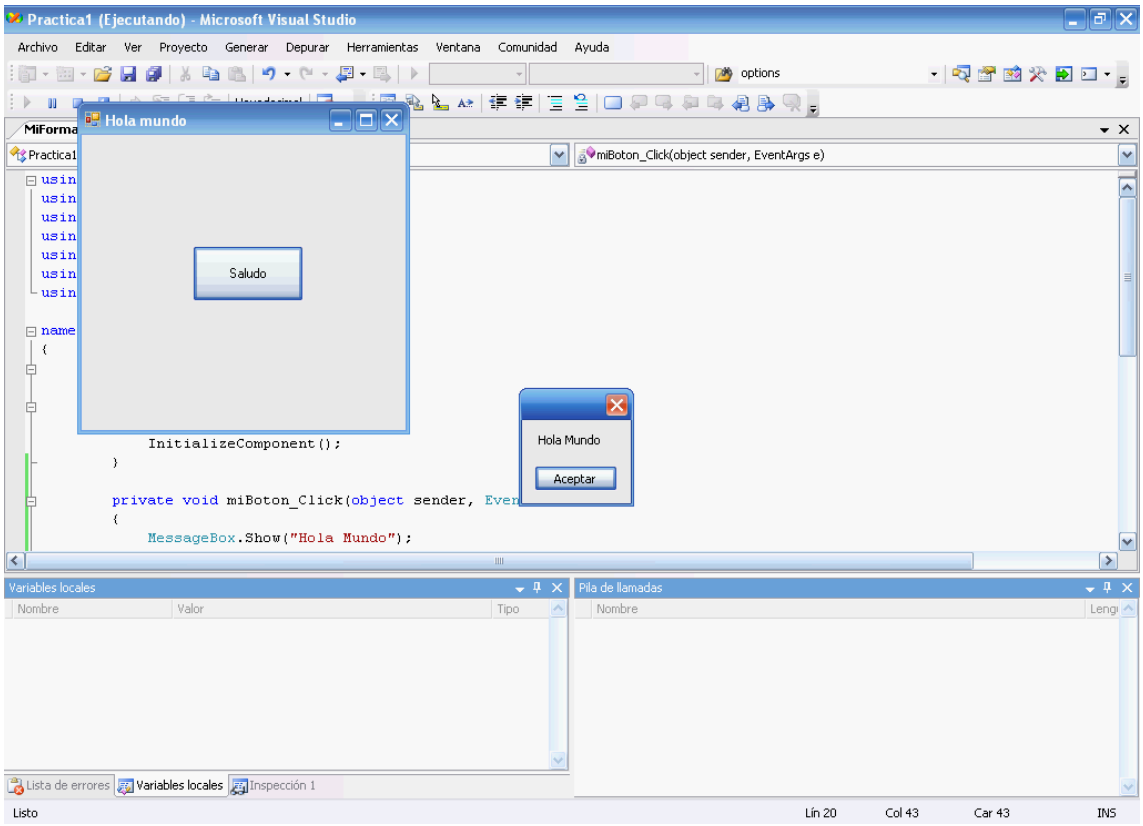


Figura P2.14

¡Tu programa funciona! Una vez que hayas terminado de probar tu mini-aplicación, detén la ejecución realizando alguna de estas acciones:

- Cierra la ventana principal (la que tiene el título “Hola Mundo”).
- Elige Depurar -> Detener depuracion, en el menu principal.
- Presiona *mayus+F5*.

Cuestionario

- ¿Que cualidades y defectos encuentras en *Visual Studio 2005*?
- El ejercicio que se realizó en esta práctica ¿te ayudó a comprender como funciona este entorno de trabajo?

6.3 Práctica 3 - Diagrama de Casos de Uso

Objetivos

- Conocer qué es un diagrama de casos de uso en UML.
- Aprender a diseñar un diagrama de casos de uso con ayuda de Microsoft Visio.

Introducción

Diagramas de Casos de uso

Los diagramas de casos de uso permiten definir la funcionalidad del sistema en función de las necesidades de los usuarios. Además establecen los límites del sistema.

A pesar de que los diagramas de casos de uso son sencillos de entender para el usuario también son muy generales, deberán expresar con claridad lo que los usuarios necesitan y de esta forma el usuario entenderá el comportamiento que tendrá el sistema. Una vez identificadas las principales funcionalidades en los casos de uso, se puede detallar cada uno. Para esto se hace un diagrama amplificado de cada caso de uso general que lo amerite.

Los elementos de los diagramas de casos de uso son:

Actores: Un actor es un rol que un usuario juega con respecto al sistema. Un actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema. Hay diferentes tipos de actores:

- **Usuarios:** Son las personas que usan el sistema.
- **Otros sistemas:** Son los sistemas con los que el sistema interactúa.

La misma persona física puede interpretar varios actores, el nombre del actor describe el papel desempeñado y tiene debajo el nombre:

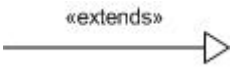
Casos de Uso: Los casos de uso son las funciones que el sistema deberá hacer para regresar un valor o servicio al actor. El proceso de desarrollo estará dirigido por los casos de uso. Se denota por un óvalo, con el nombre del caso de uso dentro:

Un actor se relaciona con un caso de uso por medio de una línea.

Un caso de uso se relaciona con otro, por medio de una flecha. La relación entre casos de uso puede ser por:

- <<uses>>: Cuando un caso de uso "usa" a otro.

- <<extends>>: Cuando un caso de uso "extiende" de otro.



Alcance del sistema: El alcance del sistema son los casos de uso que se desarrollarán para el sistema en cada ciclo. Se denota por un cuadrado que define el alcance del sistema conteniendo los casos de uso dentro el diagrama de casos de uso.

Ejemplo:

El diagrama de casos de uso en nuestro ejemplo, describirá una **aplicación simple de software para una biblioteca**. Esta es una variación acerca del ejemplo clásico para casos de uso y es suficiente para mostrar la manera en la que funciona este tipo de diagrama.

El objetivo de nuestro sistema es administrar las operaciones que se llevan a cabo en una biblioteca, se prestan y regresan diferentes medios de información, las entregas retrasadas se manejan de la siguiente forma: 3 pesos por día en retraso de DVDs, 2 pesos por día en retraso de revistas y un peso por día en retraso de libros. Un prestatario no puede sacar prestado un medio si: tiene algún retraso de entrega pendiente, si posee cinco artículos en préstamo, o si ha acumulado más de cinco pesos en multas.

A continuación se muestra el diagrama de Casos de Uso (*Figura P3.1*), que contiene la mayoría de los casos de uso de la aplicación.

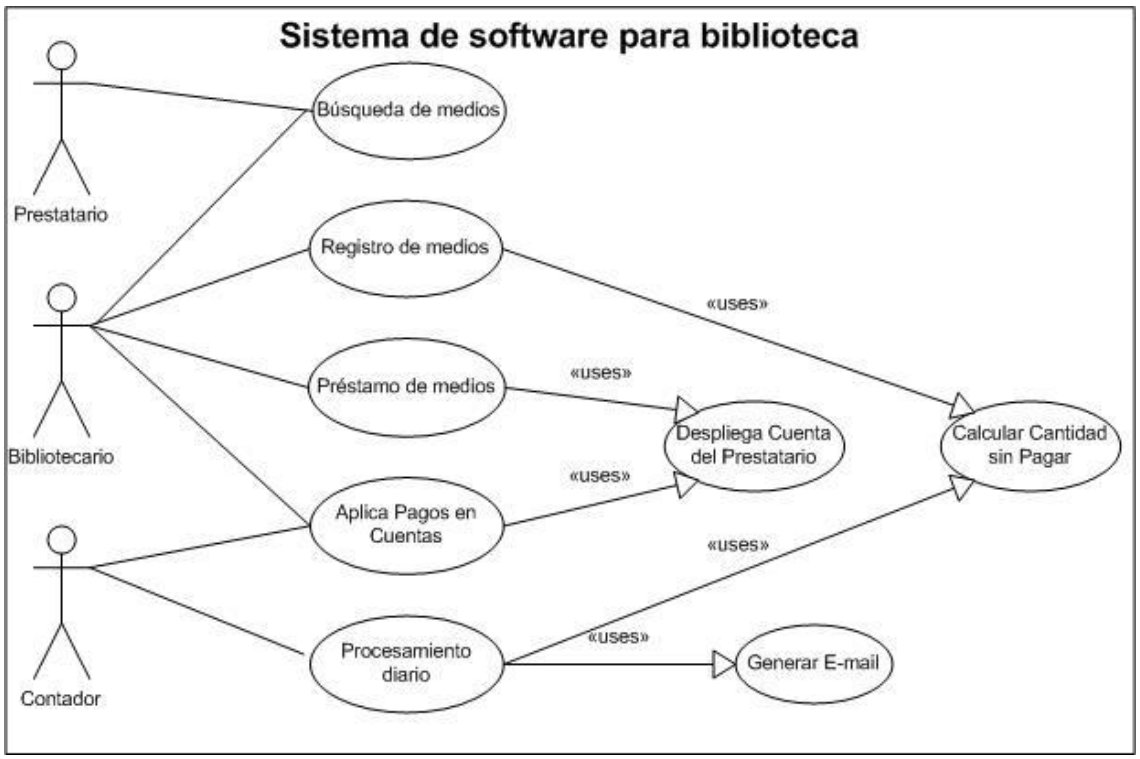


Figura P3.1

Como puedes ver en el diagrama, hay tres actores: El prestatario, el bibliotecario y el contador. El **prestatario** (como puedes imaginar) representa a la gente que pide préstamos de medios de información. En nuestra aplicación, los medios consisten en DVDs, revistas y libros. Como en la mayoría de los sistemas software de bibliotecas la única acción que se permite al prestatario es realizar **búsquedas de medios**.

El **bibliotecario** realiza búsquedas de medios, pero también realiza otras actividades. El **bibliotecario *presta*** medios de información, ***registra*** los medios de información entregados, y aplica los pagos de multas en las cuentas del **prestatario**. El **contador** realiza dos acciones principalmente "**aplica los pagos**" en las cuentas del **prestatario** y además ejecuta un "**procesamiento diario**". El procesamiento diario se ejecuta cada tarde para generar correos electrónicos que advierten al **prestatario** sobre artículos no devueltos ó multas retrasadas.

Notarás que nuestro diagrama de casos de uso, identifica algunas relaciones "usa", indicando los lugares donde una funcionalidad de la aplicación es utilizada por uno o más casos de uso. **Registro de medios** y **Procesamiento diario** *usan* el caso de uso **Calcular Cantidad sin Pagar. Préstamo de medios** y **Aplica Pagos en Cuentas** *usan* el caso de uso **Despliega Cuenta del Prestatario.** Por último **Procesamiento diario** usa **Generar E-mails**.

Desarrollo

Estos son los pasos para diseñar el diagrama mostrado anteriormente:

- Iniciamos el software Visio y creamos un nuevo *Diagrama Modelo de UML*. Agregamos 8 formas *Caso de uso* en el área de dibujo (como en la práctica 1), también agregamos 3 formas *Actor* (*Figura P3.2*).

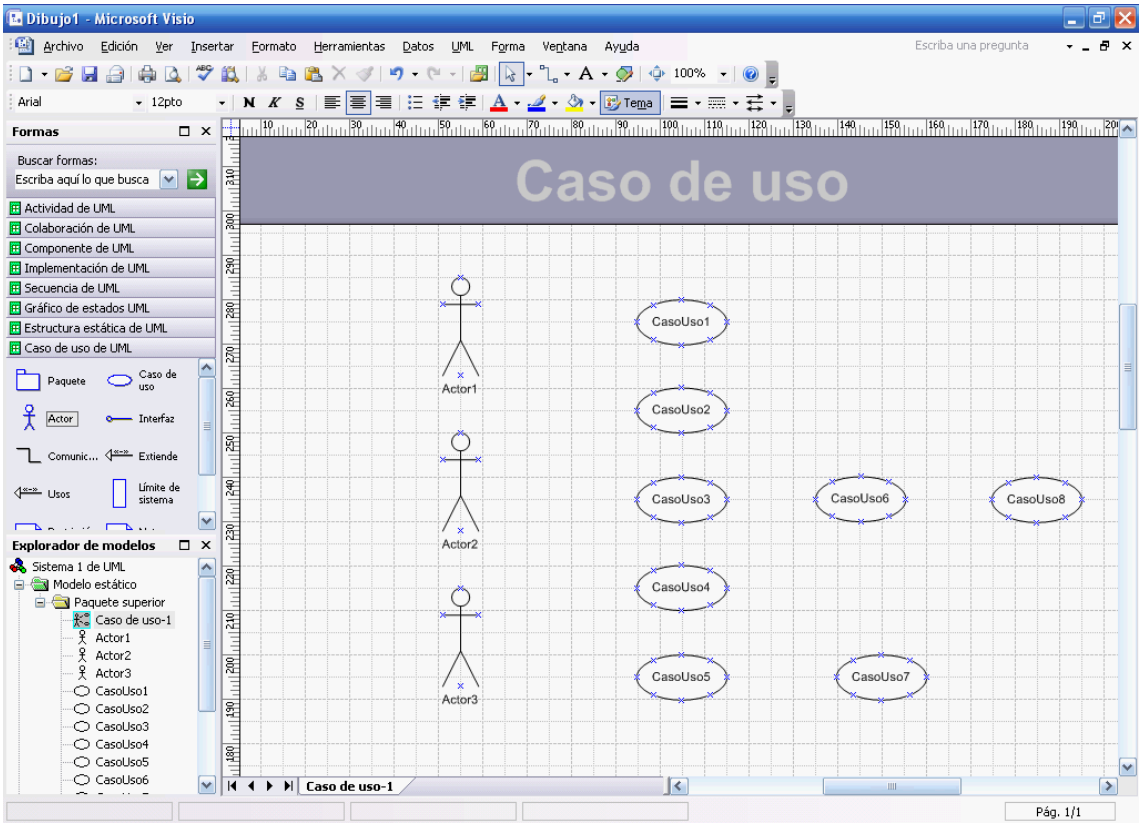


Figura P3.2

- A continuación colocamos el nombre adecuado a cada forma (como en la práctica 1) el resultado se muestra en la *Figura P3.3*.

Después conectamos los casos de uso y los actores, ayudándonos de las formas ***Comunicados*** y ***Usos*** que se encuentran en la misma plantilla **Caso de Uso UML**. Aparecerán las palabras Fin1 y Fin2 en los conectores ***comunicados,*** no es necesario colocar estos detalles en nuestro diagrama, así que vamos a borrar estas palabras mediante estos pasos:

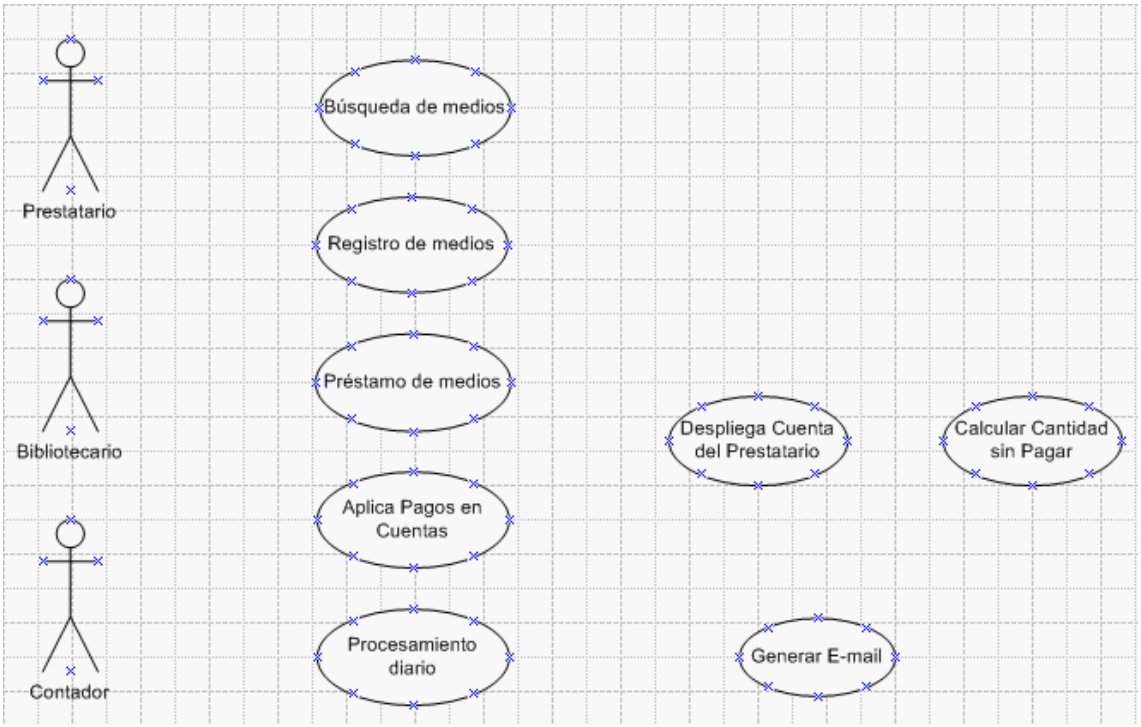


Figura P3.3

- Agregamos una forma ***comunicados*** al área de dibujo.
- Accedemos a la ventana de propiedades de la forma agregada, dando doble click en ella.
- En la sección de Extremos de asociación, da doble click en la palabra ***Fin 1*** para que sea editable y bórrala, de igual forma borra la palabra ***Fin 2*** y los asteriscos de la columna multiplicidad (*Figura P3.4*).
- Copia y pega la forma que modificaste para conectar los demás casos de uso, así no tendrás que borrar las palabras Fin1 y Fin2 cada vez.

Recuerda que el inicio y final de los conectores cambia de color verde a rojo para indicar que las formas están bien conectadas (práctica 1).

- Por último agregamos una forma *Limite de sistema* cambiamos el nombre del sistema y modificamos su tamaño para adecuarse a nuestro diagrama. El resultado final se muestra en la *Figura P3.5*.

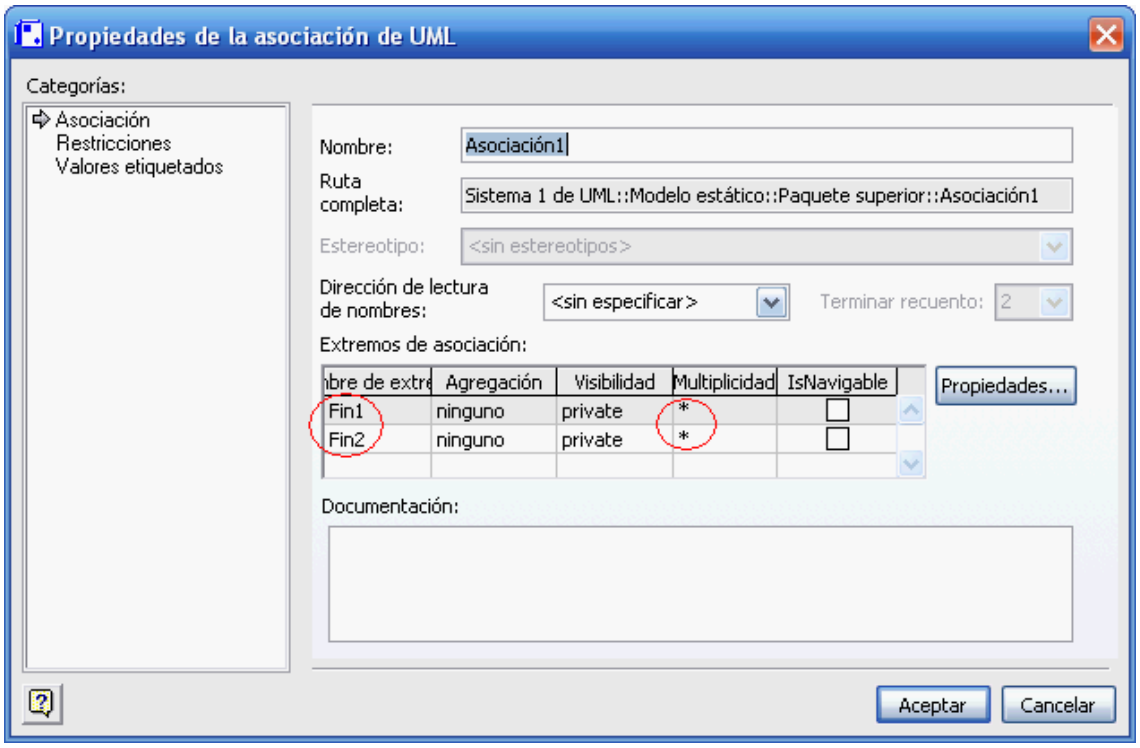


Figura P3.4

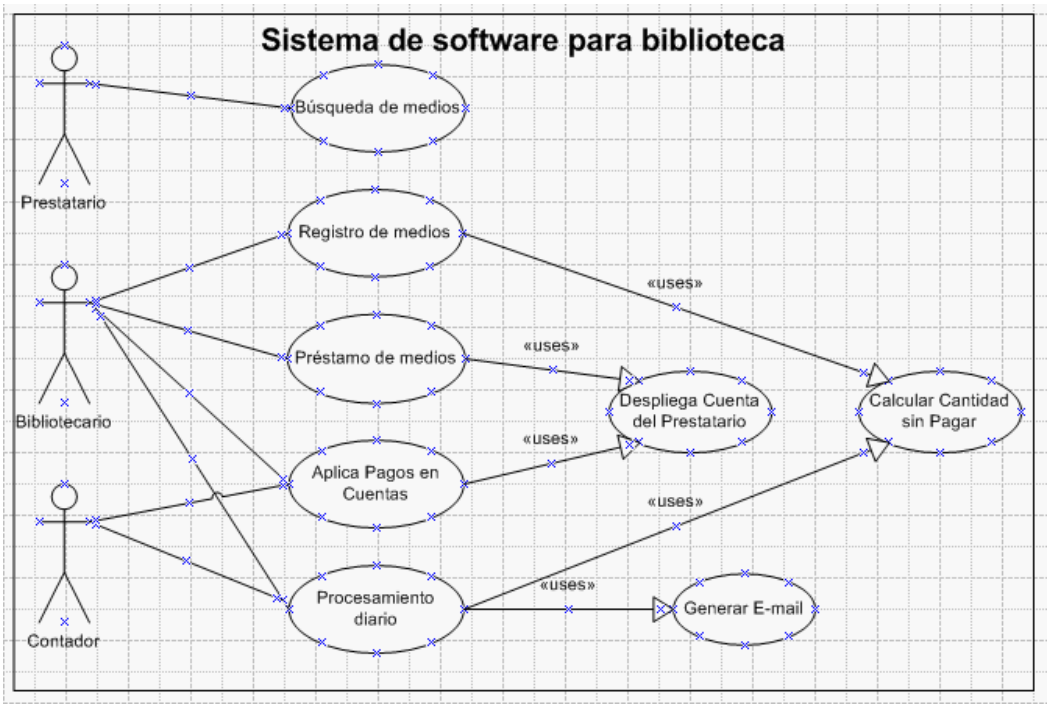


Figura P3.5

Nota: Para guardar tu diagrama como un dibujo y ponerlo en tus propios documentos (como la *Figura P3.1*), selecciona *guardar como* y elije guardar como tipo JPG (o el formato que prefieras), presiona aceptar en la siguiente ventana.

Actividades

- Recopila los casos de uso para el sistema de software que crearás durante este curso.
- Con ayuda del software *Visio*, crea un diagrama por cada caso de uso que hayas recopilado.

Cuestionario

- ¿Para qué sirven los diagramas de casos de uso?
- ¿Cuáles son los elementos de los diagramas de casos de uso?

6.4 Práctica 4 - Prototipo de interfaz de usuario

Objetivos

- Aprender a diseñar la interfaz de una aplicación Web con Visual Studio 2005.
- Aprender a interactuar con los controles de interfaz en una aplicación web.

Introducción

Debido a que el objetivo de nuestro curso es realizar una aplicación Web, se hará uso de la tecnología **ASP.NET** para desarrollar nuestro sistema. Como se mencionó en la práctica 2, la programación se realizará en el lenguaje **C#**.

ASP.NET

ASP.NET es una nueva tecnología para el desarrollo de aplicaciones en Internet, creada por Microsoft como parte del entorno de desarrollo en .NET. Esto permite que el desarrollo de las aplicaciones Web por medio de ASP.NET pueda realizarse utilizando cualquier lenguaje de programación que tenga un compilador para .Net.

Objetos en .NET: Como ya se ha explicado, C# es un lenguaje orientado a objetos, incluso los tipos básicos (int, double, etc.) derivan de alguna clase. Esto es debido a que en el ambiente de .NET cada clase deriva de una clase inicial llamada **Object**. Estas bases también se aplican en el diseño de una aplicación Web, ya que en .NET incluso una página Web es una clase. En la *Figura P4.1* se puede observar la jerarquía de herencia de la clase **Page**.

Namespaces

Como ASP.NET forma parte del entorno de desarrollo .NET, tenemos acceso a todas las herramientas que han sido construidas para el ***.NET Framework*** por medio de la **biblioteca de clases.** Esta biblioteca representa una fuente gigantesca de herramientas en forma de clases. Estas clases están organizadas en una jerarquía de **Namespaces.** Cuando queremos utilizar alguna de las características que .NET ofrece, solo tenemos que encontrar el namespace que contiene la funcionalidad deseada, y después importarlo en nuestra página ASP.NET. Una vez hecho esto, podemos utilizar las clases contenidas en ese **namespace**.

Por ejemplo, si queremos acceder a una base de datos desde nuestra página Web, tendríamos que importar el **namespace** que contiene las clases para este propósito, la cual sería **System.Data.SqlClient.** Para importar las clases contenidas en un **namespace** en C# se utiliza la instrucción **using**:

```
using System.Data.SqlClient;
```

Figura P4.1

Web Server Controls

Los controles de servidor Web pueden verse como versiones avanzadas de los controles de HTML. Los controles de servidor Web son aquellos que generan contenido para ti, debido a esto, ya no es necesario que utilices HTML, tener un buen conocimiento de HTML es útil, pero no es necesario ser un experto si se trabaja con ***Web Server Controls.***

Considera el siguiente elemento HTML de entrada, el cual crea una caja de texto:

```
<input type="text" name="usernameTextBox" size="30" />
```

El control de servidor Web equivalente es el control TextBox, y se ve así:

```
<asp:TextBox id="usernameTextBox" runat="server" Columns="30" />
```

Como puedes observar los controles de servidor Web se encierran en una etiqueta **<asp: />**, además los controles de servidor Web requieren el atributo **runat="server"** para funcionar correctamente. Existen mas controles de servidor Web que controlen HMTL, algunos ofrecen características que simplemente no están disponibles utilizando HTML.

Archivos Code-behind

Muchas de las compañías que emplean equipos de desarrollo, usualmente dividen el proyecto en dos grupos: diseño visual y desarrollo funcional. Debido a que generalmente los ingenieros de software son malos diseñadores y los diseñadores son malos ingenieros de software. El problema con varías tecnologías de desarrollo de aplicaciones Web es que el código se encuentra escrito directamente en las páginas Web. Esto provoca que la división del desarrollo de un sistema sea muy difícil, debido a que no existe una separación entre los elementos visuales y los elementos lógicos. En respuesta a estos problemas, es que ASP.NET introduce una nueva forma de desarrollo, que permite trabajar de forma separada a los diseñadores de la interfaz y a los desarrolladores de código. Este método llamado **code-behind,** mantiene todos los elementos de interfaz (controles Web) dentro de un archivo **.aspx** y todo el código que implementa la parte lógica del sistema se separa en una clase dentro de un archivo **code-behind** (que en nuestro caso será **.cs** porque utilizaremos C#).

Ejemplo

Esta es una sencilla aplicación la cual contiene tres controles Web: una etiqueta, una caja de texto y un botón. Cuando se presiona el botón, el contenido de la caja de texto se asigna al texto de la etiqueta.

Primero analicemos el código del archivo que se encarga de la interfaz (**Prueba.aspx**):

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Prueba.aspx.cs"
Inherits="Prueba" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
        <head runat="server">
                <title>Página sin título</title>
        </head>
        <body>
                <form id="form1" runat="server">
                        <div>
                        Nombre:
                        <asp:TextBox ID="name" runat="server"></asp:TextBox>
                        <asp:Button ID="submitButton" runat="server" Text="Aceptar"
                        OnClick="submitButton_Click" /><br />
                        <asp:Label ID="messageLabel" runat="server"></asp:Label></div>
                </form>
        </body>
</html>
```

La primera línea marcada en negritas indica:

- El código **“<%@ Page %>”** indica que esta página es una página ASP.NET porque utiliza la clase **Page**. Es similar a importar el namespace de la clase **Page** con: **using System.Web.UI.Page**.
- **Language="C#"** indica que el lenguaje que se utiliza es C#.
- **CodeFile="Prueba.aspx.cs"** indica el nombre del archivo code-behind: **Prueba.aspx.cs**
- **Inherits="Prueba"** indica el nombre de la clase contenida en el archivo code-behind.

El siguiente bloque de código en negritas define los controles Web. Observa que se encuentran dentro de un tag **<form>** con el atributo **runat="server"**. Los controles Web tienen un **ID** por medio del cual se puede hacer referencia a ellos dentro del código. Observa que al control Web **Button** se le agregó el atributo **OnClick**, esto indica que el método **submitButton_Click** se ejecutará cada vez que se presione el botón.

Ahora analicemos el código del archivo code-behind (**Prueba.aspx.cs)**:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Prueba : System.Web.UI.Page
{
   protected void submitButton_Click(object sender, EventArgs e)
   {
      messageLabel.Text = name.Text;
   }
}
```

Lo primero que se puede observar es que se importan varios **Namespaces**, estos son los que se ponen por omisión en una aplicación Web. Lo realmente importante esta marcado en negritas:

- El código **public partial class Prueba : System.Web.UI.Page** crea una clase parcial llamada **Prueba**, la cual extiende a la clase **System.Web.Ui.Page.**

- Después se crea el método del evento **OnClick** que pertenece al control Web **Buttton,** el método se crea con el código: **protected void submitButton_Click(object sender, EventArgs e).**
  Finalmente se utilizan los IDs de los controles Web para asignar el contenido de la caja de texto a la etiqueta (**messageLabel.Text = name.Text**;).

¿Qué es una clase parcial?

Las clases parciales son una nueva característica de .NET 2.0. Una clase parcial permite a una clase esparcirse sobre múltiples archivos. ASP.NET utiliza esta característica para facilitar la vida a los programadores. Nosotros escribimos una parte de la clase en el archivo code-behind, y ASP.NET genera la otra parte de la clase por nosotros. Agregando las declaraciones de objetos para cada elemento de la interfaz del usuario. Como se puede observar en nuestro método **submitButton_Click,** se utilizan los objetos de los controles Web **messageLabel** y **name** sin tener que crear instancias de las clases **Label** y **TextBox**.

El resultado de ejecutar nuestra pequeña aplicación Web se muestra en la *Figura P4.2.*

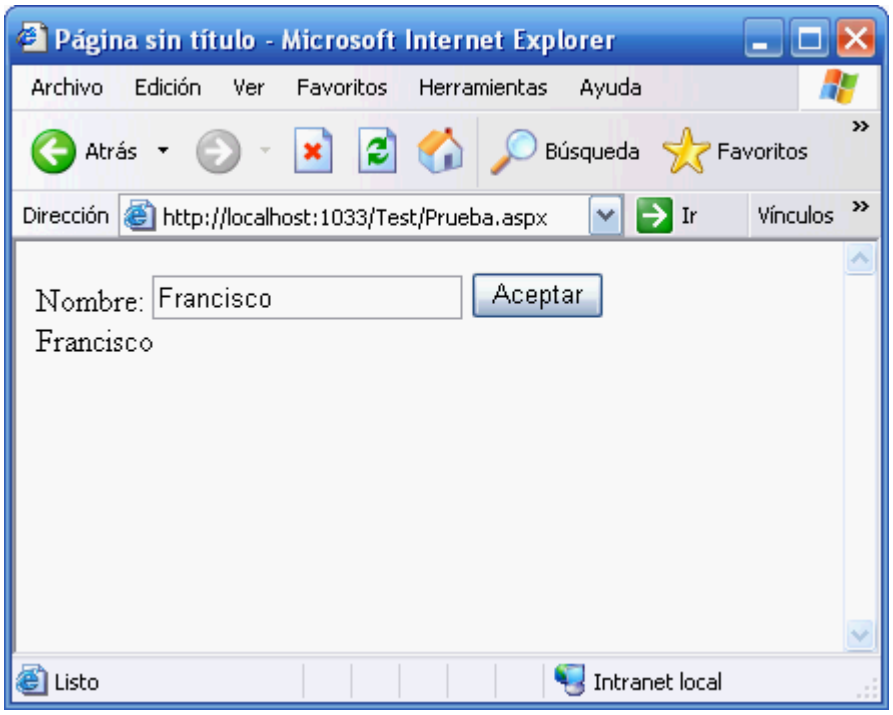


Figura P4.2

Desarrollo

- Para crear una nueva aplicación Web en Visual Studio 2005, selecciona **Archivo-> Nuevo -> Sitio Web** en el menú principal o selecciona crear sitio Web en la página de inicio.

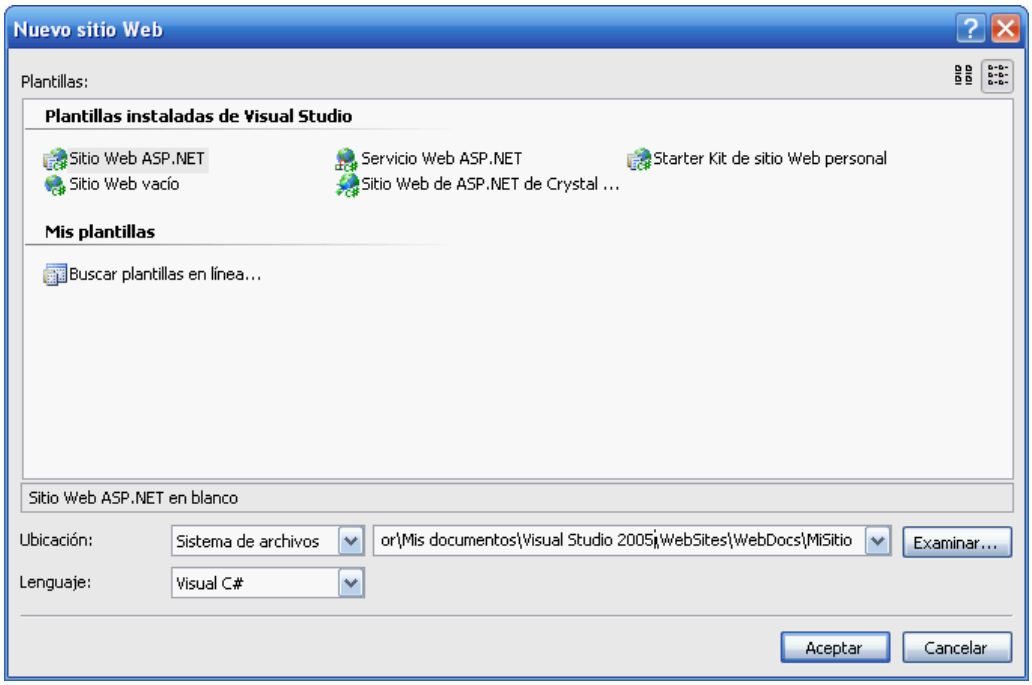


Figura P4.3

Aparecerá una ventana como la de la *Figura P4.3*. Como puedes observar Visual Studio 2005 ofrece varias plantillas de sitios Web. Selecciona la primer plantilla **Sitio Web ASP.NET**. En el combo de selección de lenguaje elije **Visual C#,** y en ubicación, introduce el nombre de la carpeta que contendrá todos los archivos de tu sitio Web.

La interfaz de Visual Studio que se mostrará debe ser similar a la de la *Figura P4.4.*

Como podrás notarlo la interfaz es muy similar a la interfaz que se presenta cuando creas una nueva aplicación de Windows (Práctica 2). Se presentan las mismas ventanas: el cuadro de herramientas, el explorador de soluciones y la ventana de propiedades. Sin embargo, existen diferencias importantes.

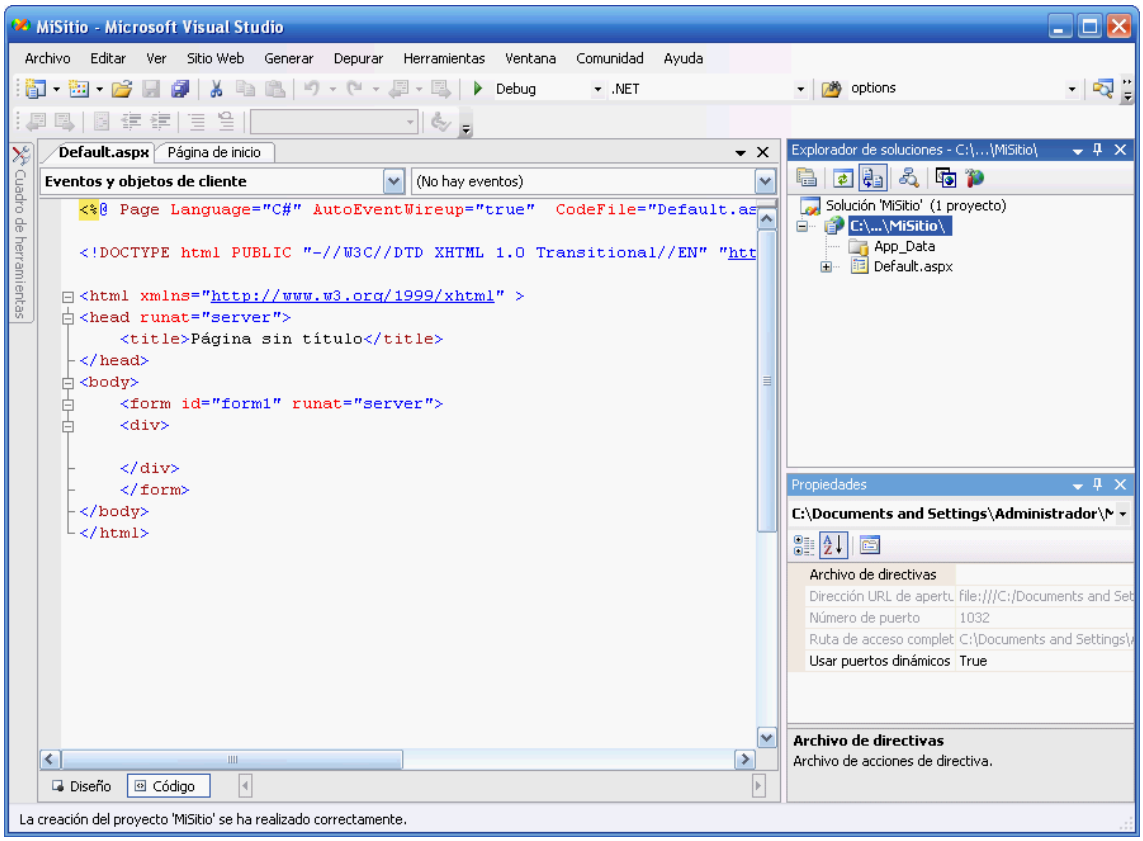


Figura P4.4

En el explorador de soluciones (Figura P4.5) se presentan dos nodos en el árbol:

- **App_Data** es un folder especial que ASP.NET usa para almacenar los archivos de la base de datos.
- **Default.aspx** es la forma Web que Visual Studio crea por omisión (default) cuando creas un nuevo sitio Web. Al expandir ese nodo, encontrarás un archivo code-behind llamado **Default.aspx.cs.** Este archivo está asociado a la forma Web dentro del código. Si le cambias el nombre a la forma **Default.aspx** (click derecho sobre el nodo y seleccionar cambiar nombre), el archivo **code-behind** también cambiará de nombre, al igual que su referencia en el código.

El cuadro de herramientas también cambia un poco, porque ahora puedes agregar controles Web en lugar de agregar controles para formas de Windows.

Visual Studio 2005, ofrece una herramienta muy poderosa para diseñar formas Web de una manera muy rápida. Para acceder al **diseñador de formas Web** presiona el botón de la esquina inferior izquierda llamado Diseño. La interfaz que se presenta debe ser similar a la *Figura P4.6*. El diseñador de formas tiene una interfaz **WYSIWYG** (**What you see is what you get**) que es similar a la de Dreamweaver, FrontPage y otros.

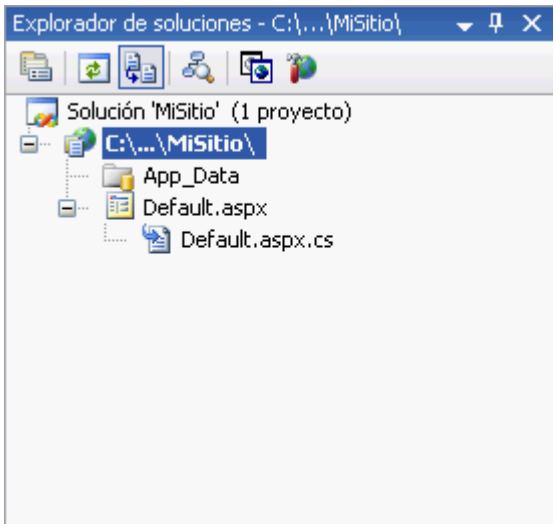


Figura P4.5

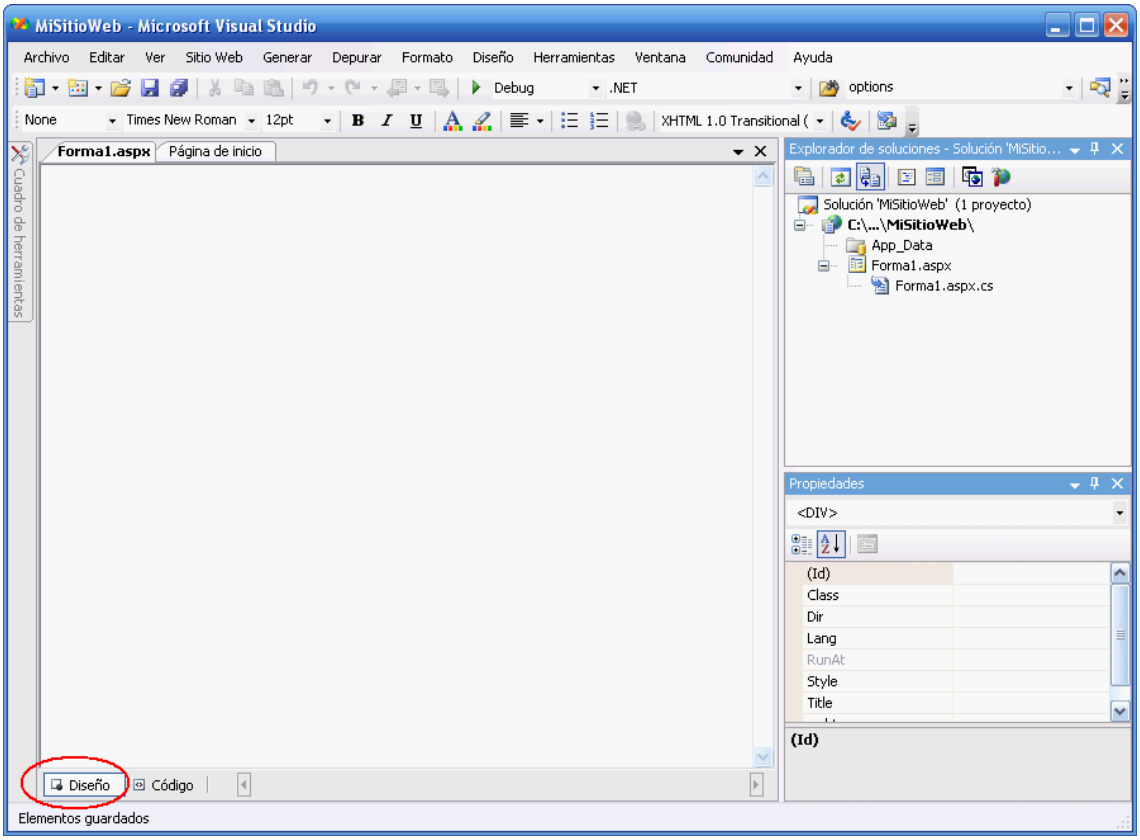


Figura P4.6

Diseñando una pequeña aplicación Web

Para mostrar las capacidades del diseñador de formas Web vamos a diseñar una muy pequeña aplicación Web. Para ser exactos, diseñaremos la aplicación Web que utilizamos en el ejemplo de esta misma práctica.

Ten en cuenta que la mayoría de las acciones que se presentan a continuación son casi iguales a las acciones que realizaste en la práctica 2, por lo tanto se explicarán con menos detalle.

- Crea un proyecto nuevo como mencionamos anteriormente, o regresa al editor de código del proyecto, si ya habías creado uno (click en el botón código que está a un lado del botón diseño para ir al editor de código). Cambia el nombre de la forma **Default.aspx** en el explorador de soluciones, ponle el nombre **Forma1**, el nombre del archivo **code-behind** también debe cambiar al igual que su referencia en el código (*Figura P4.7*).

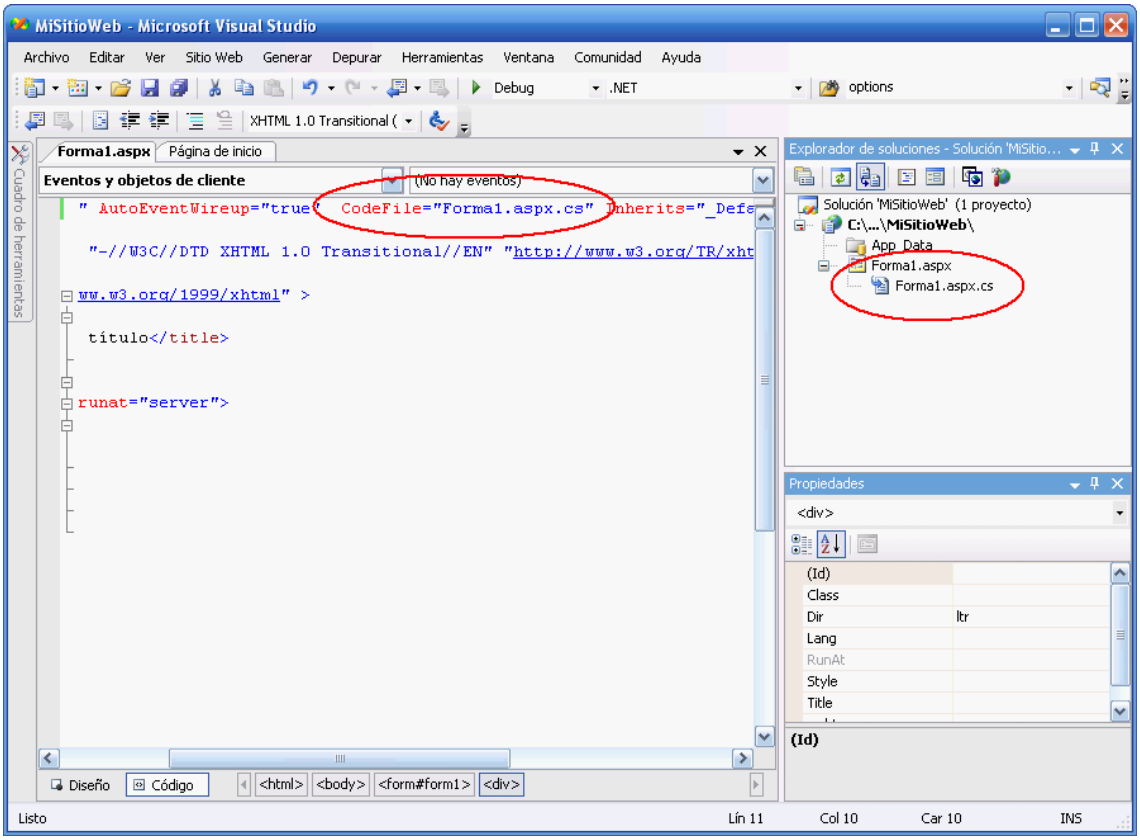


Figura P4.7

- Cambia a la interfaz del Diseñador de formas Web y escribe la palabra “Nombre:” al inicio de la página.
- Arrastra un control Web **TextBox** desde el cuadro de herramientas hasta el área de diseño de la página (Puedes agregar controles de tres maneras distintas como en la práctica 2).
- De la misma manera agrega un control **Button** y un control **Label**

- Tu página debe verse similar a la de la *Figura P4.8.*

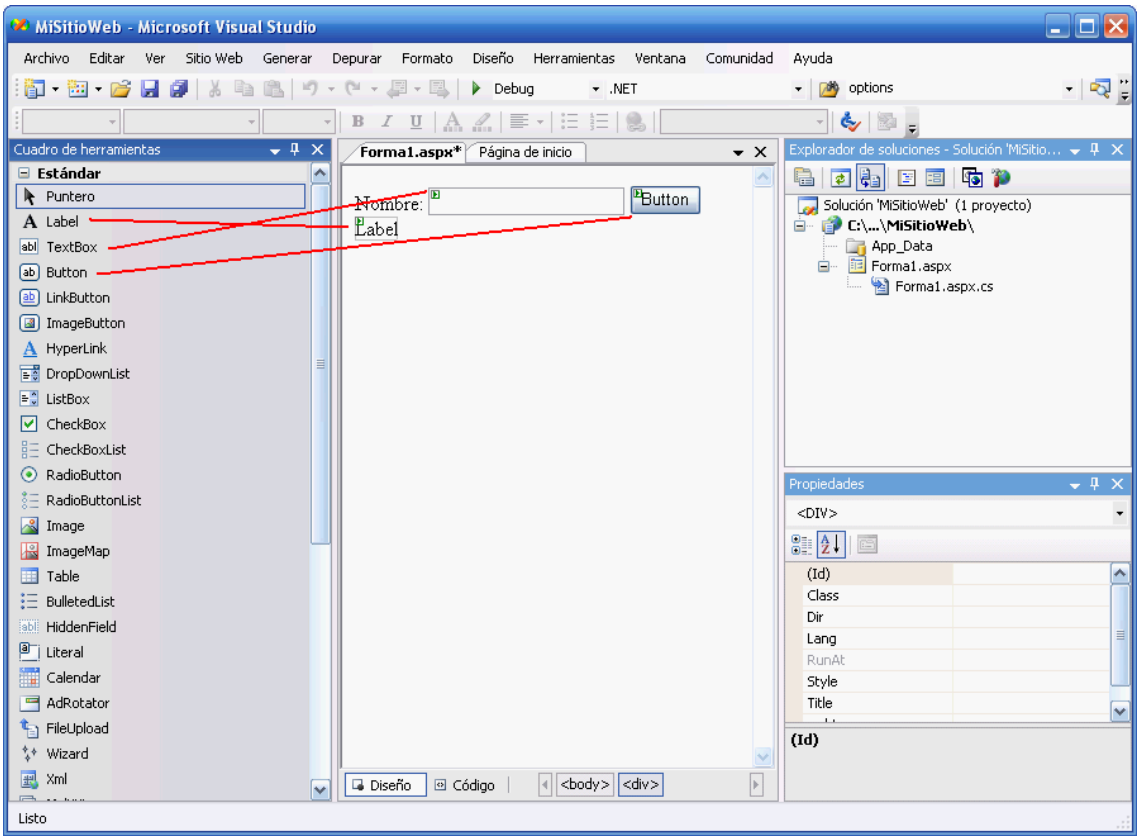


Figura P4.8

- Selecciona la caja de texto y cambia su nombre en la ventana de propiedades, borra el nombre TextBox1 y escribe MiCaja. Cambia el nombre del botón a MiBoton y el nombre del control Label a MiEtiqueta. Observa los cambios en el código (*Figura P4.9*).

- Regresa a la vista de diseño de formas Web y da doble click sobre el botón. Automáticamente se abrirá el código del archivo **code-behind** (Forma1.aspx.cs) y se creará el método correspondiente al evento principal del botón (Click). Escribe el siguiente código dentro del método: **MiEtiqueta.Text = MiCaja.Text;**

- Cambia el nombre de la clase parcial **_Default** por el nombre **MiClase**. Tu código debe verse igual al de la *Figura P4.10*.

```
e Language="C#" AutoEventWireup="true"  CodeFile="Forma1.aspx.cs" I

PE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www

mlns="http://www.w3.org/1999/xhtml" >
unat="server">
tle>Página sin título</title>


rm id="form1" runat="server">
v>
 Nombre:
 <asp:TextBox ID="MiCaja" runat="server"></asp:TextBox>
 <asp:Button ID="MiBoton" runat="server" Text="Button" />
 <br />
 <asp:Label ID="MiEtiqueta" runat="server" Text="Label"></asp:Label
orm>
```

Figura P4.9

- Para que tu aplicación funcione correctamente también debes sustituir el nombre **_Default** por **MiClase** en la sentencia **Inherits** que se encuentra en la primera línea del código en el archivo de la página MiForma.aspx (*Figura P4.11*)

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class MiClase : System.Web.UI.Page
{
    protected void MiBoton_Click(object sender, EventArgs e)
    {
        MiEtiqueta.Text = MiCaja.Text;
    }
}
```

Figura P4.10

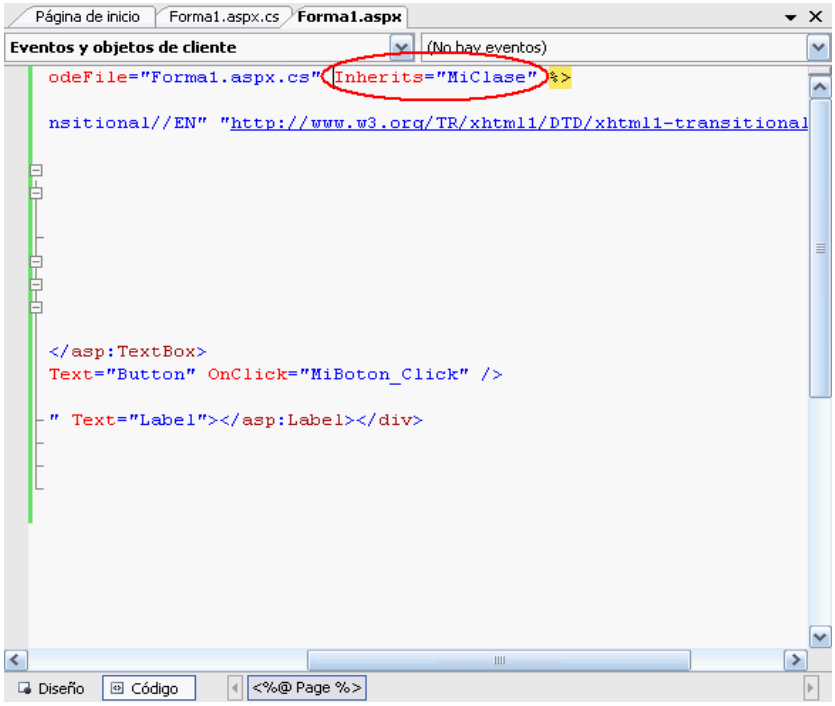


Figura P4.11

- Regresa a la vista diseño de formas Web y selecciona el botón, cambia su propiedad **Text** en la ventana de propiedades y escribe **Aceptar**. Selecciona la etiqueta y deja en blanco su propiedad **Text** (en la ventana de propiedades). Tu página debe verse similar a la de la *Figura P4.12*.

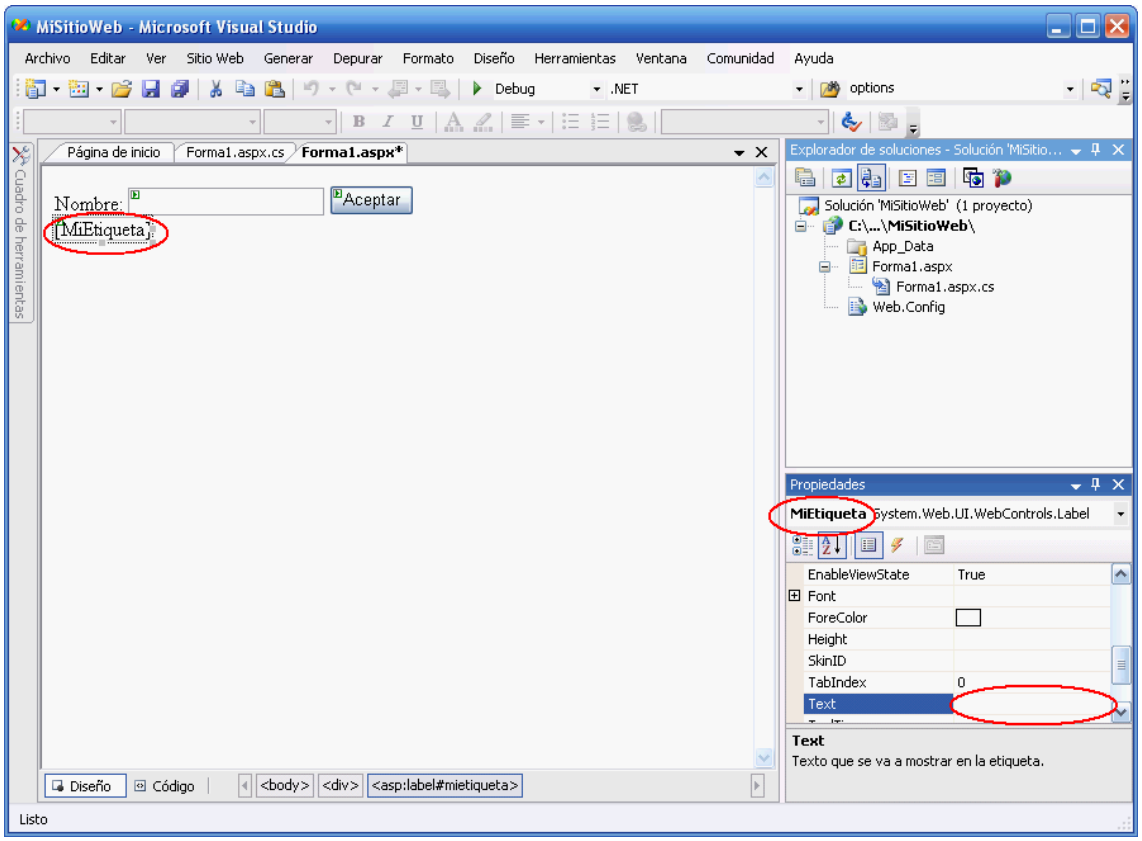


Figura P4.12

- Presiona F5 para ejecutar tu aplicación, una ventana parecida a la de la *Figura P4.13* aparecerá. Selecciona la primera opción y presiona aceptar.

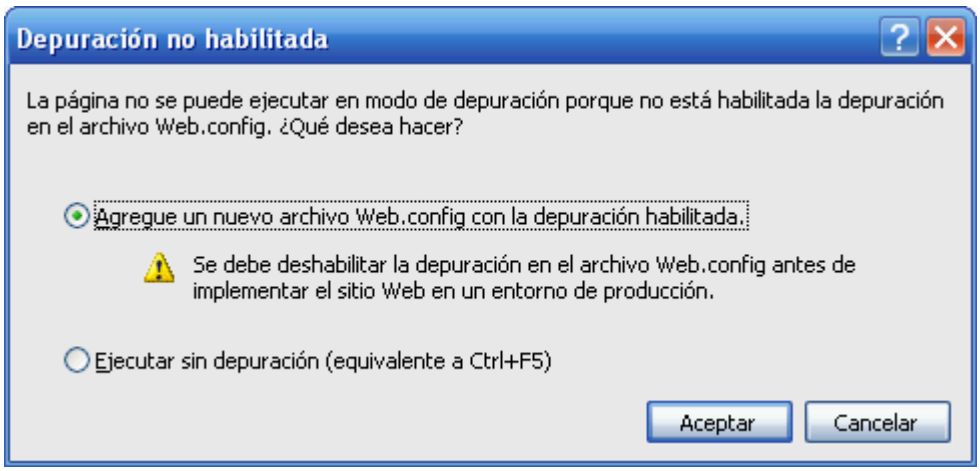


Figura P4.13

Nota: Una verdadera aplicación Web no cuenta con una sola clase y una página (forma). Cuando sea necesario tener más páginas o clases, selecciona sitio **Web->Agregar nuevo elemento** o presiona **Ctrl+Shift+A**, aparecerá un cuadro de dialogo en el que podrás seleccionar que elemento quieres agregar.

Tu aplicación debe verse como la de la *Figura P4.14*. El texto escrito en la caja de texto aparecerá en la línea de abajo (en la etiqueta) cada vez que presiones el botón **Aceptar**.

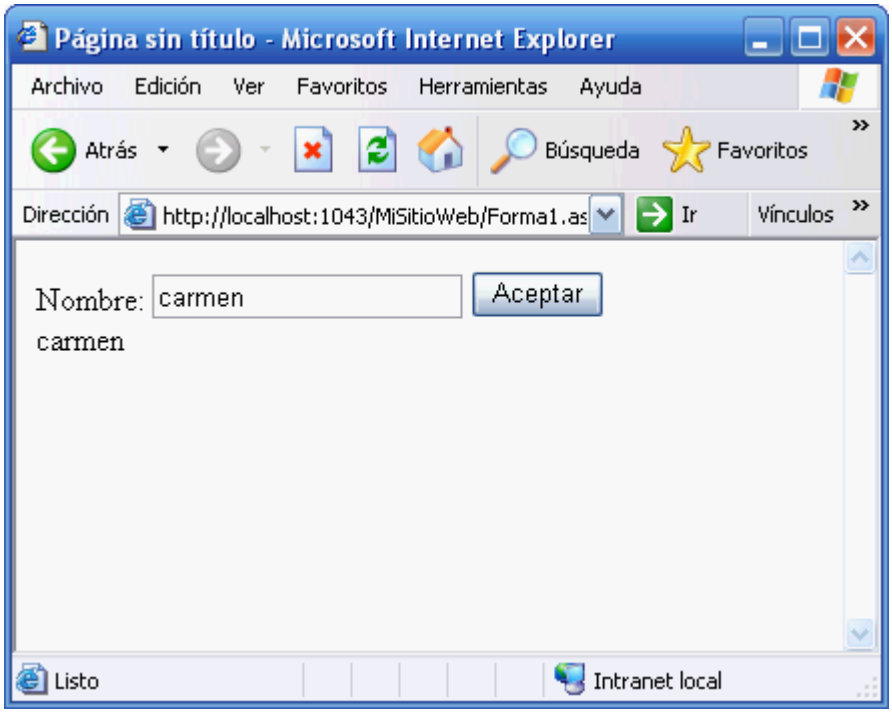


Figura P4.14

Actividades

- Definir cómo será la interfaz de tu sistema web, cuantas páginas habrá, que formularios contendrá cada una y como se navega entre ellas.
- Crear la interfaz de usuario de tu sistema web, ayudándote del diseñador de formas web que Visual Studio proporciona.
- Ligar las pantallas mediante links, de tal forma que la navegación entre ellas sea continua.

Cuestionario

- ¿Que caracteristicas debe tener la interfaz de una aplicación web?
- ¿Cuales son los controladores de interfaz web que consideras más importantes?

6.5 Práctica 5 - Diagrama de Clases

Objetivos

- Conocer qué es un diagrama de clases en UML.
- Aprender a diseñar un diagrama de clases con ayuda de Microsoft Visio.

Introducción

Diagramas de clases

En estos diagramas se definen las características de cada una de las clases del sistema, así como sus relaciones. A continuación vemos los elementos de un diagrama de clases:

Clase: Está representada por un rectángulo que dispone de tres apartados:

- **Nombre:** Cada clase tiene un nombre único, que la identifica entre las demás.
- **Atributos:** Representa las propiedades o atributos de la clase, que se encuentran en todas las instancias de la clase. Pueden indicarse en el diagrama mostrando su nombre, su nombre y su tipo, e incluso su valor por defecto.
- **Operaciones (Métodos):** Representan los métodos de la clase que se encuentran en todas las instancias de la misma. Se indican en el diagrama mostrando el nombre del método, sus parámetros y el tipo de valor que regresa (en caso de que el método regrese un valor).

Nota: Para indicar el nivel de protección que tienen los atributos y métodos se escribe alguno de los siguientes símbolos:

Símbolo	Nivel de Protección
+	Public
-	Private
#	Protected

En la *Figura P5.1* se muestra el ejemplo de una clase diagramada en UML. La clase con nombre ***ManejaBaseDeDatos***, contiene tres atributos:

- **Servidor:** Cadena que almacena el nombre del servidor.
- **Usuario:** Cadena que almacena el nombre del usuario de la base de datos.
- **Clave**: Cadena que almacena la clave del usuario de la base de datos.

También dispone de tres métodos:

- **conecta:** Recibe tres cadenas que contienen el nombre del servidor, el nombre del usuario y la clave del usuario.
- **hazConsulta:** Recibe una cadena que contiene la consulta que desea realizarse y regresa una cadena que contiene el resultado de la consulta.

- **cierra:** Cierra la base de datos.

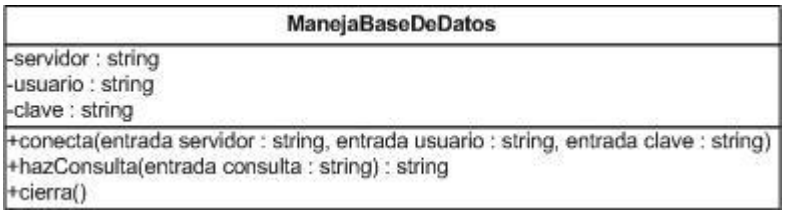
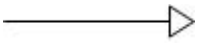


Figura P5.1

Relaciones entre clases: Las clases pueden relacionarse entre sí de varias formas. En las relaciones de clases, la clase origen es desde la que se realiza la acción de relacionar, es decir, desde la que parte la flecha y la clase destino es la que recibe la flecha.

- **Dependencia:** Es una relación de uso, es decir una clase usa a otra que la necesita para su cometido. Se representa con una flecha discontinua va desde la clase que utiliza a la clase utilizada. Con la dependencia mostramos que un cambio en la clase que utiliza puede afectar al funcionamiento de la clase que la utiliza, pero no al contrario.

- **Herencia:** Es la generalización o especialización, donde tenemos una o varias clases padre o superclase, y una o varias clases hijas o subclases.



- **Asociación:** Especifica que los objetos de una clase están relacionados con los elementos de otra clase.

- **Composición:** Una clase está compuesta por otras o una clase es parte de otra

- **Multiplicidad:** Cardinalidad de la relación, es decir, cuántos objetos de esa clase pueden participar en la relación.

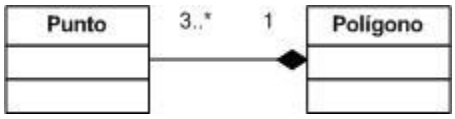


Figura P5.2. Ejemplo de composición y multiplicidad

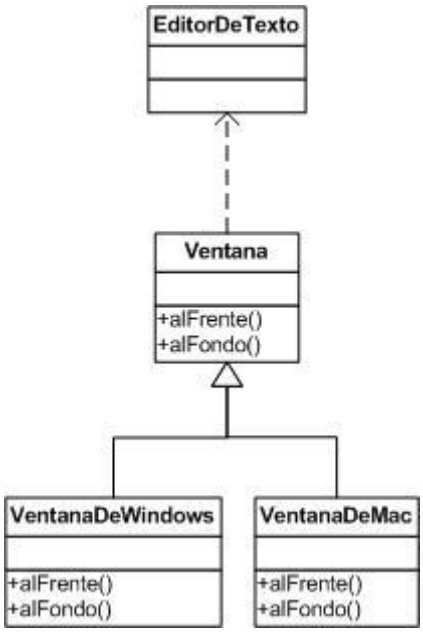


Figura P5.3 Ejemplo de Herencia y dependencia

Desarrollo

A continuación se mostrará la manera en la que se puede utilizar el software Visio, para construir un diagrama de Clases.

Primero se inicia el software Visio y se crea un nuevo *Diagrama Modelo de UML*. Esto creará un paquete superior en la rama modelo estático del explorador de modelos (*Figura P5.4*).

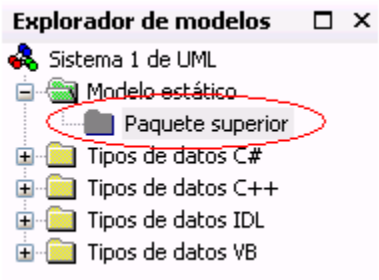


Figura P5.4

Se utilizará el ejemplo de la *Figura P5.1* para mostrar cómo se crea el dibujo que representa una clase.

- Da clic-derecho en el ***paquete superior*** y selecciona ***nuevo->Diagrama de estructura estática*** (un diagrama de clase es un tipo de diagrama de estructura estática). Esto crea un nuevo diagrama de estructura estática y lo abre en el área de dibujo de Visio.
- Renombra el diagrama de clase a ***Mi Diagrama*** dando click-derecho sobre el nodo Estructura estática-1.
- Agrega una forma ***clase,*** arrastrándola desde la plantilla ***Estructura estática de UML*** (en la ventana ***Formas***) hacia el área de dibujo.
- Accede a la ventana de sus propiedades (doble click sobre la clase) y cambia su nombre (*Figura P5.5*).

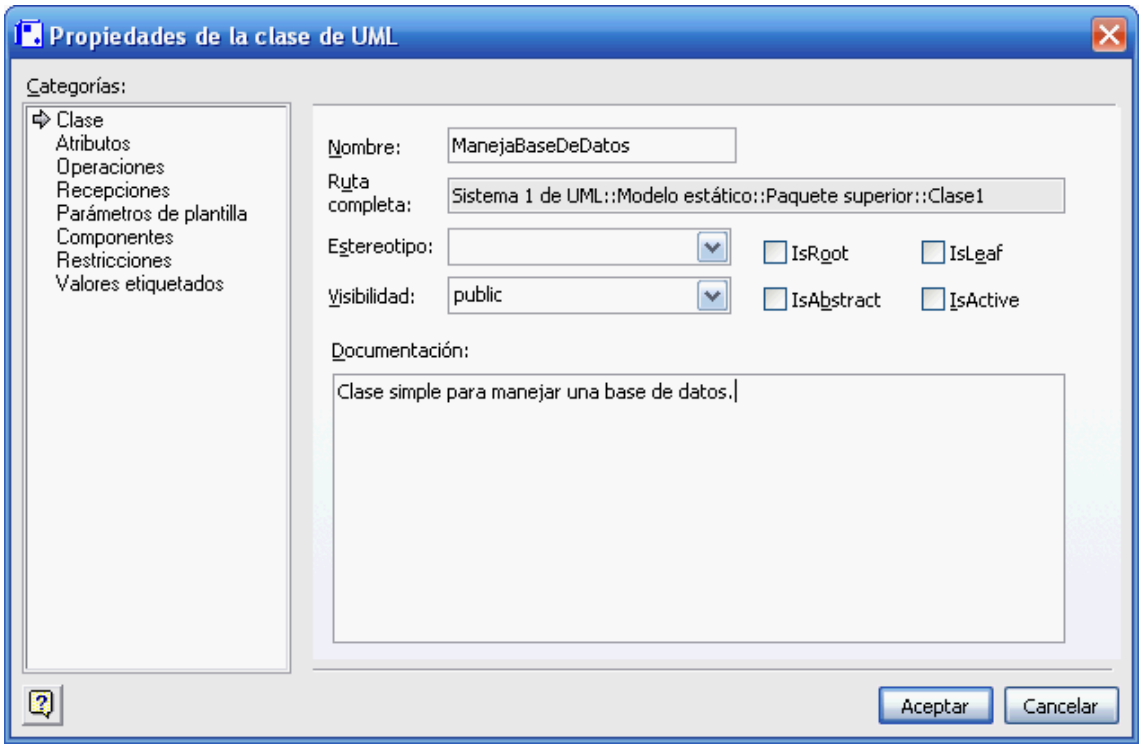


Figura P5.5

- En la categoría ***Atributos*** se agrega un atributo presionando el botón nuevo, selecciona el nuevo atributo y presiona el botón propiedades (*Figura P5.6*).

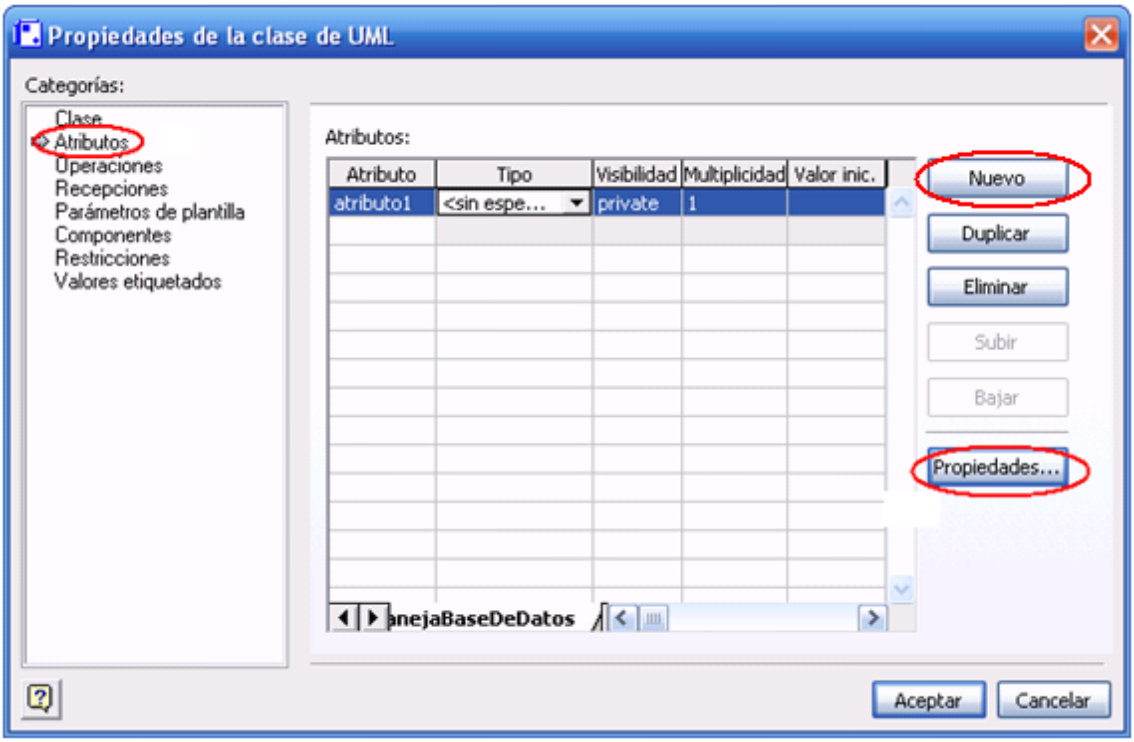


Figura P5.6

- Aparecerá la ventana de propiedades del atributo. Cambia los atributos nombre, tipo y visibilidad como en la *Figura P5.7*.

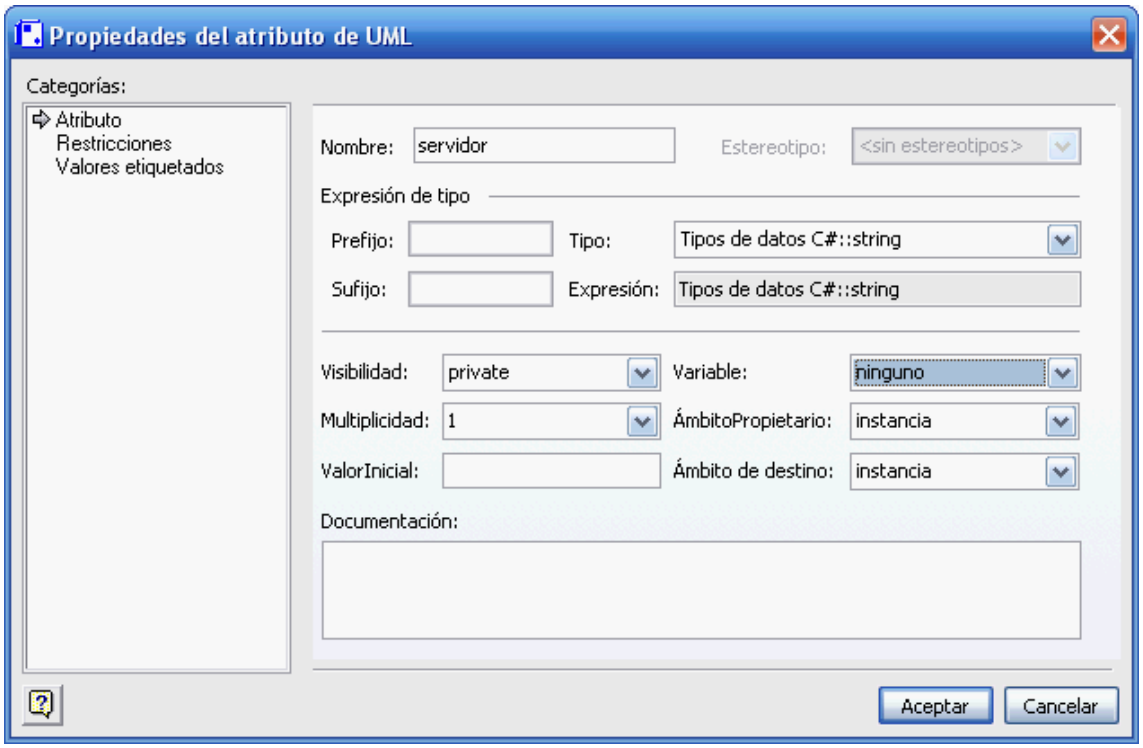


Figura P5.7

- Agrega los otros dos atributos de la misma manera.
- En la categoría ***operaciones*** puedes agregar un nuevo método presionando el botón ***nueva.*** Selecciona el nuevo método y presiona el botón ***propiedades*** (*Figura P5.8*).

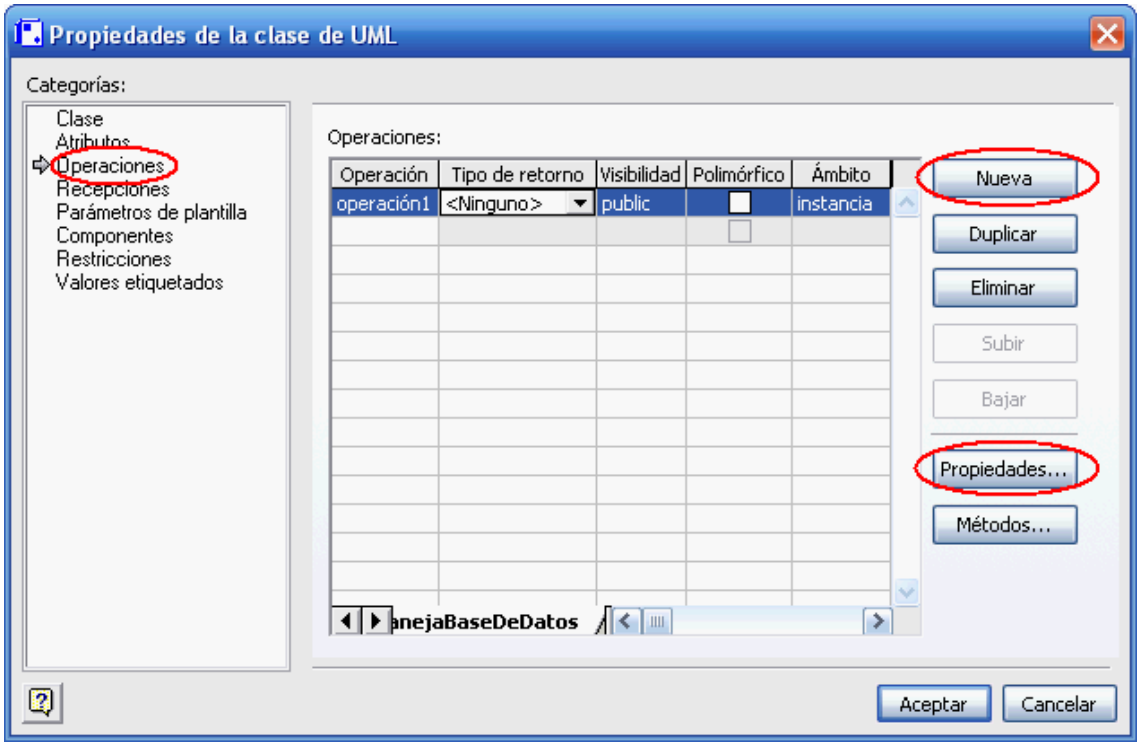


Figura P5.8

- Aparecerá la ventana de propiedades de la operación (método), cambia los atributos nombre, tipo y visibilidad como en la *Figura P5.9*. Después selecciona la categoría parámetros y agrega los tres parámetros que recibe el método de la misma manera en que agregaste los atributos de la clase.

- Agrega los dos métodos restantes de forma similar a la que utilizaste para agregar el método **conecta**.

- El resultado final deberá ser parecido a la *Figura P5.10*. Nota como se ha modificado el árbol del ***explorador de modelos***.

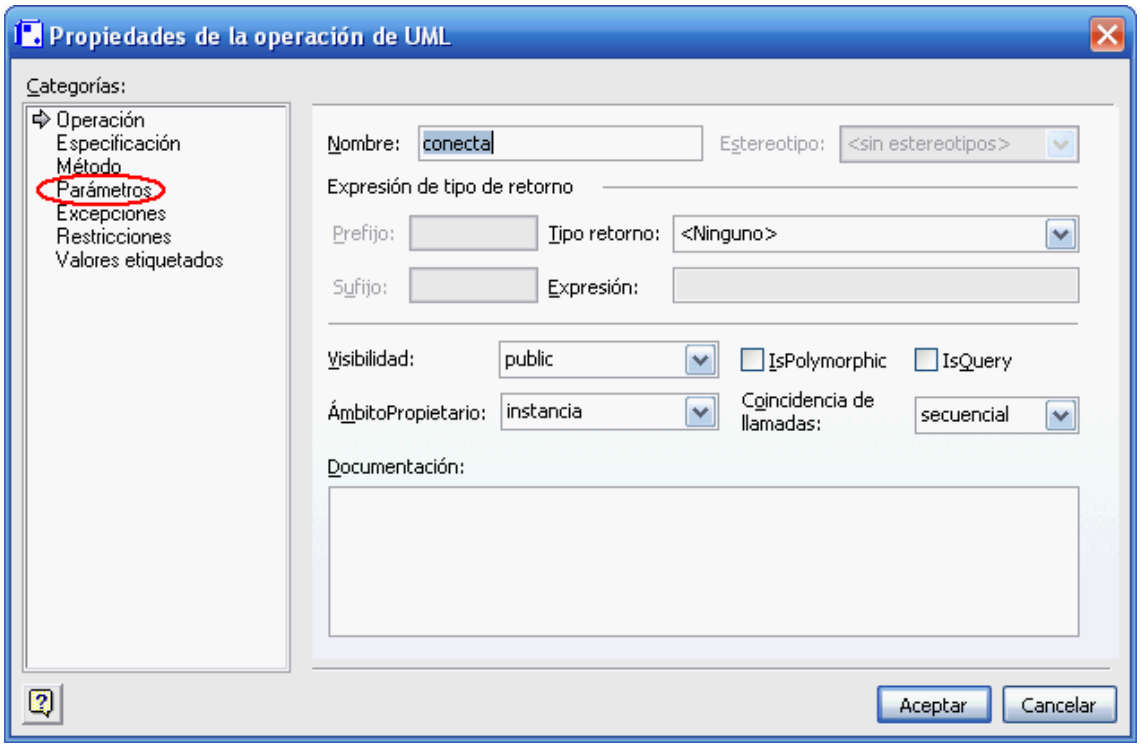


Figura P5.9

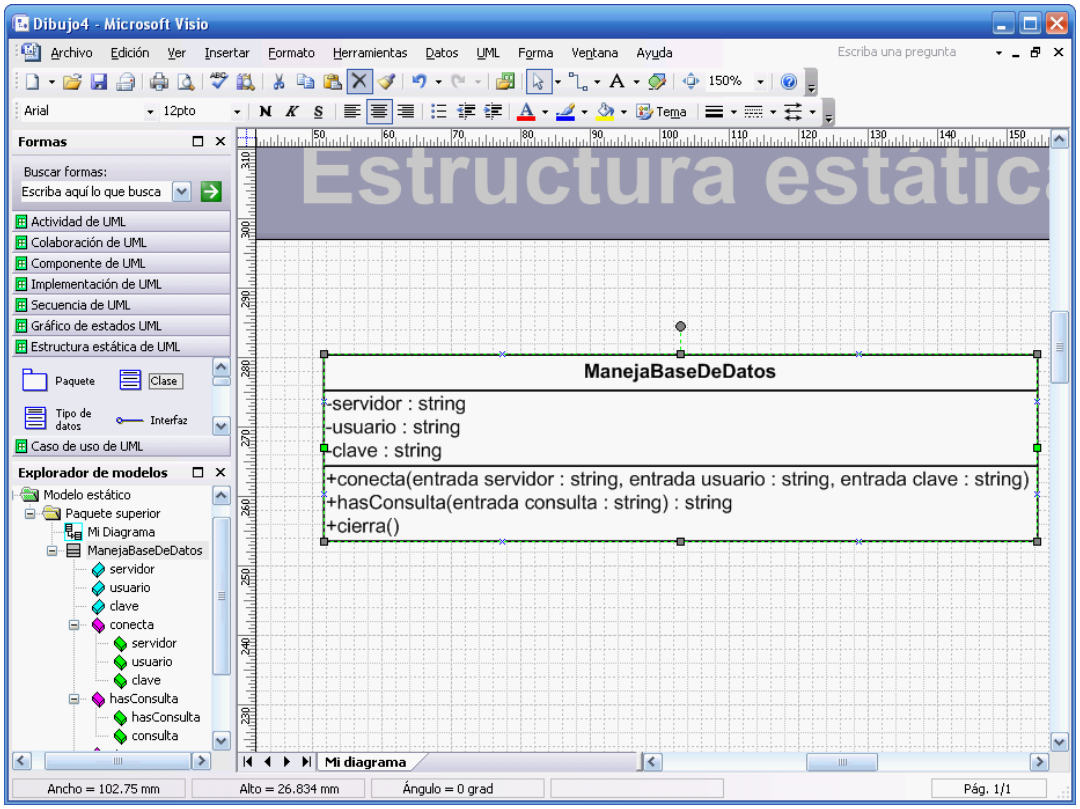


Figura P5.10

Los conectores que se presentan en la plantilla ***estructura estática de UML*** permiten describir todas las relaciones que existen en los diagramas de clase. Las habilidades que se han adquirido durante esta práctica, así como en las prácticas 1, 3, son suficientes para que puedas realizar tu propio diagrama de clases.

Actividades

- De acuerdo a tu diagrama de casos de uso, divide las funcionalidades que tendrá tu sistema de software y agrúpalas en las clases que conformarán el sistema. Recuerda incluir las *clases parciales* que se encargarán de leer los datos de los formularios que creaste en la práctica anterior.

- Realiza un bosquejo de las clases, así como de sus propiedades y sus métodos. Recuerda detallar su nivel de protección.

- Dibuja el diagrama de clases con el software Visio.

Cuestionario

- ¿Para qué sirven los diagramas de clases?
- ¿Cuáles son los elementos de los diagramas de clases?

6. 6 Práctica 6 - Diagrama de Secuencia

Objetivos

- Conocer qué es un diagrama de secuencia en UML.
- Aprender a diseñar un diagrama de secuencia con ayuda de Microsoft Visio.

Introducción

Diagrama de secuencia

El diagrama de secuencia es un diagrama para modelar interacción entre objetos en un sistema. Se hace un diagrama de secuencia para el comportamiento normal de cada caso de uso y uno para el comportamiento excepcional. Este diagrama incluye los objetos de las clases que se usan para implementar un escenario y los mensajes que se envían entre los objetos.

Se debe examinar la descripción del caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se tiene modelada la descripción del caso de uso como una secuencia de varios pasos, entonces puedes recorrer esos pasos para descubrir qué objetos son necesarios para que se pueda ejecutar cada paso, apoyándote en el diagrama de clases.

Los conceptos más importantes de los diagramas de secuencia son:

Actor: Es el elemento que inicia las acciones del caso de uso enviando un mensaje a una clase de interfaz identificada para este escenario.

Objetos o instancias de las clases: Son los otros elementos del encabezado. Se representan como un rectángulo con el nombre del objeto y el de su clase, en un formato nombreObjeto: nombreClase o solo la clase a la que pertenece ese objeto. Se inicia con un objeto de una clase de interfaz. Enseguida se pone uno o más objetos de las clases de la capa de la lógica y, si es necesario, un objeto de la clase de la capa de almacenamiento.

Línea de vida de un objeto o lifeline: La línea de vida de un objeto representa la vida del objeto durante la interacción. Se representa como una línea vertical punteada.

Activación: Muestra el período de tiempo en el cual el objeto se encuentra desarrollando alguna operación. Se denota como un rectángulo delgado sobre la línea de vida del objeto.

Mensaje: El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.

En la *Figura P6.1* se pueden observar los elementos que conforman un diagrama de secuencia. Esta imagen solo es para ilustrar los conceptos vistos anteriormente.

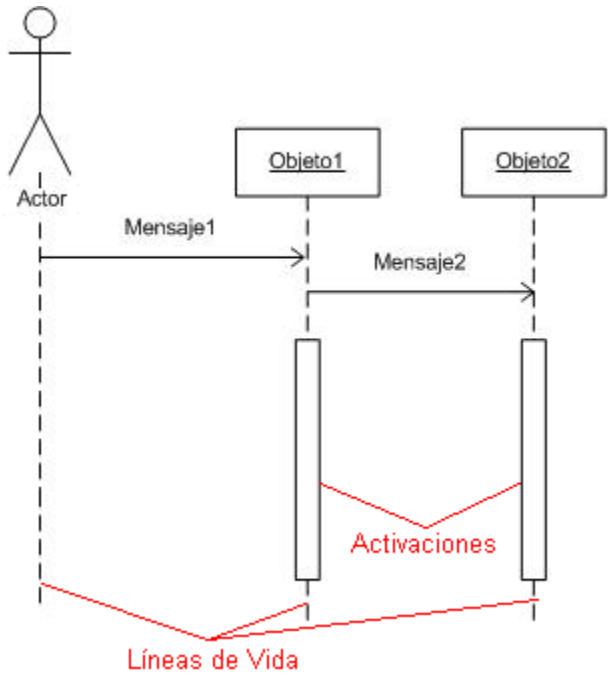


Figura P6.1.

Ejemplo

Como se mencionó anteriormente, los diagramas de secuencia se crean a partir de los casos de uso. Por cada caso de uso se puede crear un diagrama de secuencia. Vamos a retomar el ejemplo del ***sistema simple de biblioteca*** que utilizamos en la práctica 3.

Para ejemplificar el funcionamiento de los diagramas de secuencia, vamos a realizar el diagrama de secuencia correspondiente al caso de uso ***préstamo de medios***.

Crear un diagrama de secuencia es un proceso dinámico que involucra los siguientes pasos:

- Revisar los pasos que se llevan a cabo en el detalle del caso de uso.
- Se identifica el actor del caso de uso y se pone en el encabezado del diagrama de secuencia. Se pone su línea de vida.
- Crear los objetos (instancias de clase) que intervienen en el caso de uso y se ponen en el encabezado.
- Agregar los objetos al diagrama de secuencia y decidir que acciones llevará a cabo cada objeto.
- Agregar mensajes entre los objetos en el diagrama. Un mensaje enviado a un objeto en un diagrama de secuencia, crea una nueva operación en el objeto receptor.

Como puedes observar, lo primero que debemos hacer es especificar los pasos que se realizan en el caso de uso ***Préstamo de medios***:

1. El bibliotecario introduce la clave del prestatario. Esto lo hace manualmente o escaneando una credencial de biblioteca.
2. El sistema responde desplegando todos los datos de la cuenta del prestatario.
3. El bibliotecario introduce la clave del medio de información que se prestará (libro, revista o DVD). Esto también lo realiza manualmente o escaneando el código de barras.
4. El sistema responde marcando el medio de información como ***prestado***. Este paso puede ser repetido dependiendo del número de objetos que el prestatario solicite para el préstamo.

Desarrollo

Para comenzar, lo que debemos hacer es crear un nuevo diagrama de secuencia en Visio:

- Crea un nuevo diagrama de modelo de UML (como en prácticas pasadas).
- En el explorador de modelos selecciona el paquete superior y crea un nuevo diagrama de secuencia (el procedimiento es similar al de la práctica 6).

Es buena idea cambiar la orientación de la página de dibujo en Visio, ya que los diagramas de secuencia son más extensos en el eje horizontal. Para cambiar la orientación elije ***Archivo -> Configurar página*** en el menú principal y en la ventana que aparecerá selecciona el radio botón con la palabra ***horizontal***. En este momento, la interfaz de Visio debería mostrarte algo similar a la *Figura P6.2*:

A continuación agregaremos un actor (el bibliotecario) al diagrama de secuencia:

- Primero agrega un paquete ***Caso de Uso*** para que contenga al actor *bibliotecario.* Hazlo dando clic-derecho en el ***paquete superior*** dentro del ***explorador de modelos*** y selecciona ***nuevo->paquete*** y nómbralo ***Caso de Uso*** (en la ventana de propiedades que aparecerá).
- Da clic-derecho en el paquete ***Caso de Uso*** y selecciona ***nuevo->Actor***, nuevamente aparecerá una ventana de propiedades, nombra el actor como ***Bibliotecario***.
- Si realizaste correctamente estos pasos, deberías poder ver al actor en el explorador de modelos (el actor todavía no está visible en el área de dibujo, *Figura P6.3*).

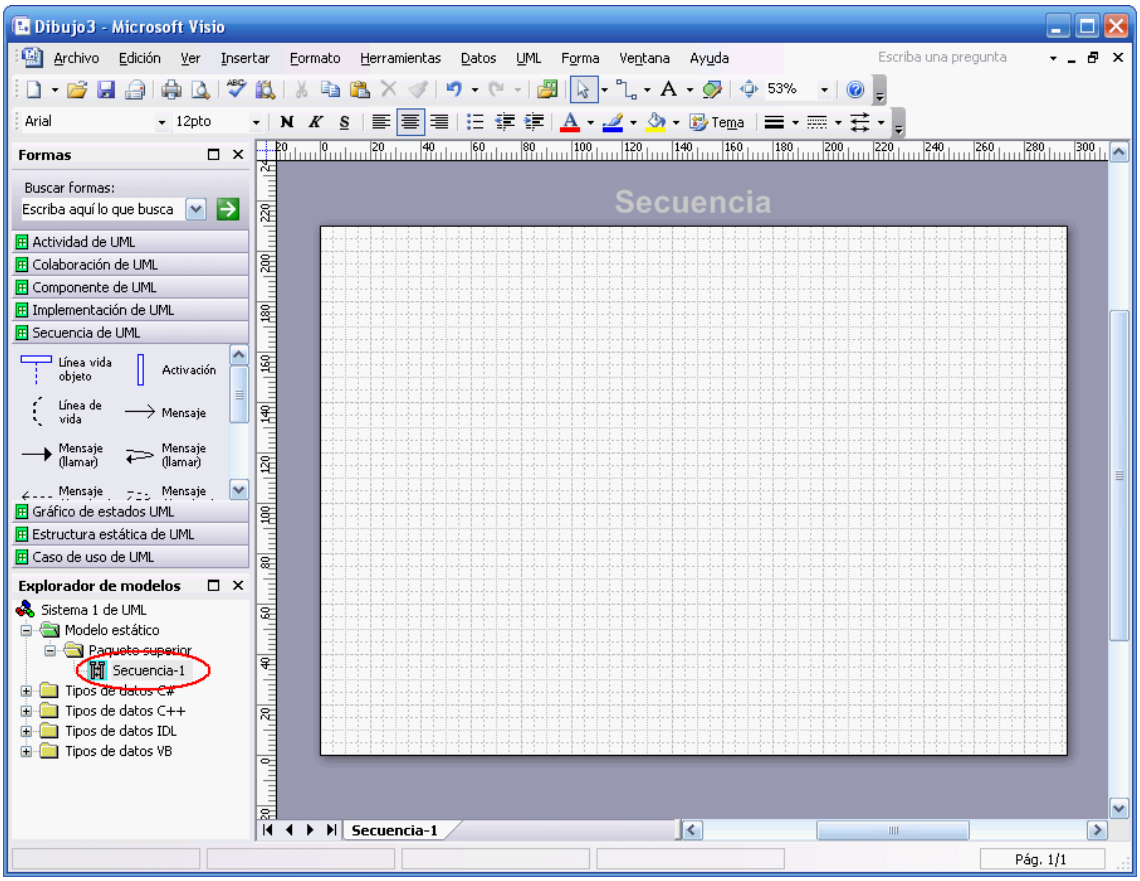


Figura P6.2

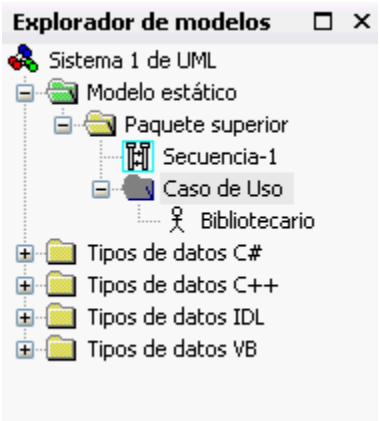


Figura P6.3

- Ahora agrega una forma ***Línea vida objeto*** al diagrama, arrastrándola desde la plantilla Secuencia de UML al área de dibujo (*Figura P6.4*).

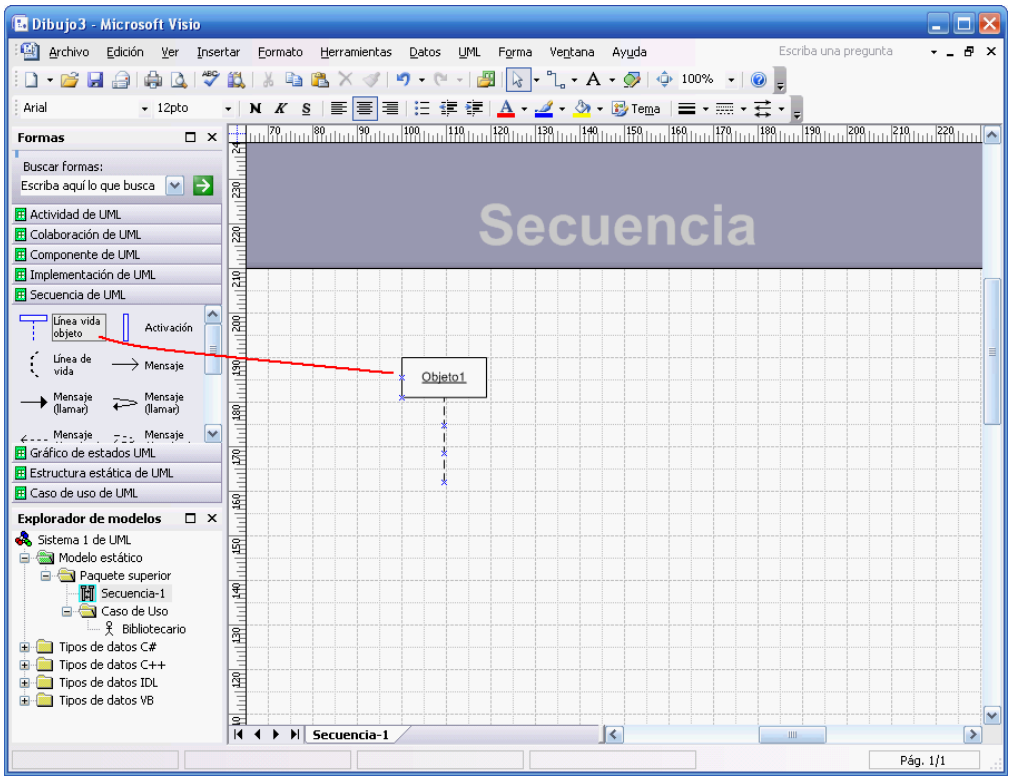


Figura P6.4

- Da doble click sobre la forma que agregaste para entrar a sus propiedades. En el combo box **Clasificador** selecciona **Caso de Uso:: Bibliotecario**, esto desplegará la forma del actor en el diagrama.

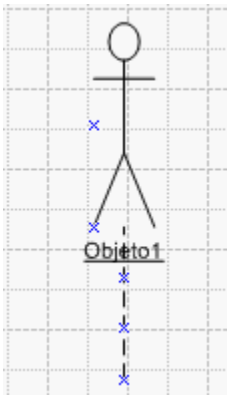


Figura P6.5

- Observa que el nombre del actor es Objeto 1 (*Figura P6.5*). Para cambiarle el nombre, da click derecho sobre el actor y selecciona opciones de presentación de formas, se desplegará una ventana como la que se muestra en la *Figura P6.6*, quita la marca de la opción Nombre, pon una marca en la opción Nombre de Clasificador y presiona aceptar.

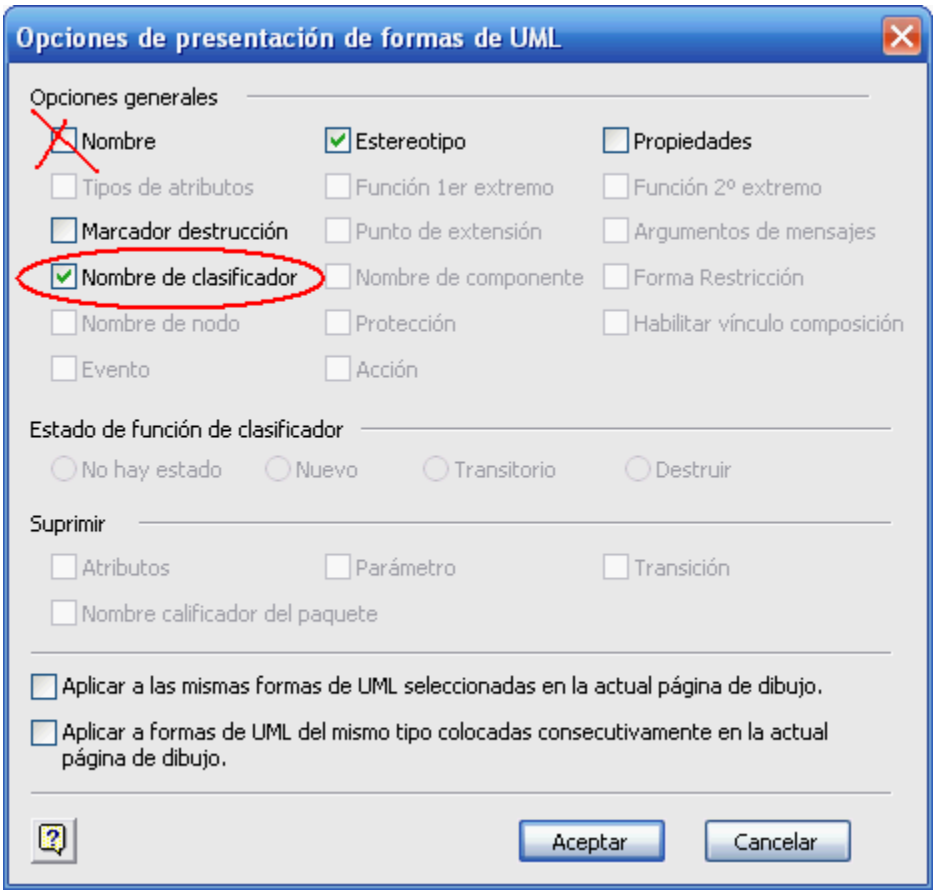


Figura P6.6

- Esto causa que el nombre del actor aparezca correctamente (*Figura P6.7*).

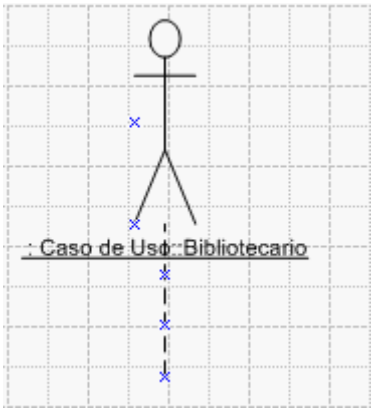


Figura P6.7

- Ahora necesitamos agregar un objeto que represente la interfaz de usuario (UI) del sistema. Los pasos que se siguen son análogos a los que seguimos para agregar al actor: Da click derecho sobre el ***paquete superior*** y elije N***ueva->Clase*** asígnale el nombre **UI** y presiona aceptar, arrastra una forma ***Línea vida objeto*** al diagrama, accede a sus

propiedades y en el combo box ***clasificador*** selecciona ***Paquete superior :: UI***. Da click derecho sobre la forma, selecciona ***opciones de presentación de forma*** y selecciona ***nombre de Clasificador*** en lugar de ***nombre***. El resultado se muestra en la *Figura P6.8*.

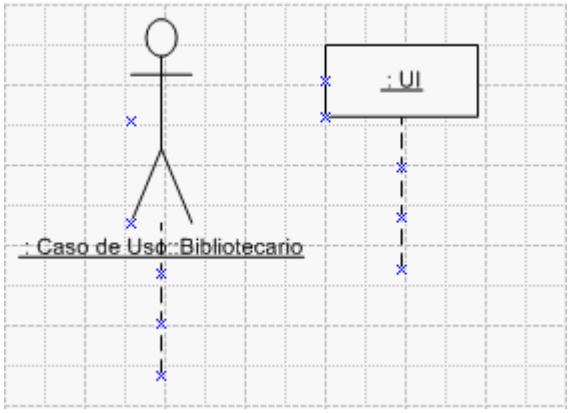


Figura P6.8

Es tiempo de agregar el primer mensaje en nuestro diagrama.

- Alarga las líneas de vida del bibliotecario y el objeto UI, después arrastra una forma ***activación*** y colócala sobre la línea de vida del bibliotecario, haz lo mismo para el objeto UI. Después arrastra una forma ***Mensaje (llamar)*** al área de dibujo y une al bibliotecario con el objeto UI. Recuerda que cuando una forma está unida correctamente a otra, sus extremos se ponen color rojo. El resultado debería ser similar al de la *Figura P6.9*.

Nota: Si te cuesta trabajo hacer que las formas queden bien alineadas, selecciona ***Herramientas->Ajustar y Pegar*** en el menú principal y deshabilita la opción ***Ajustar***).

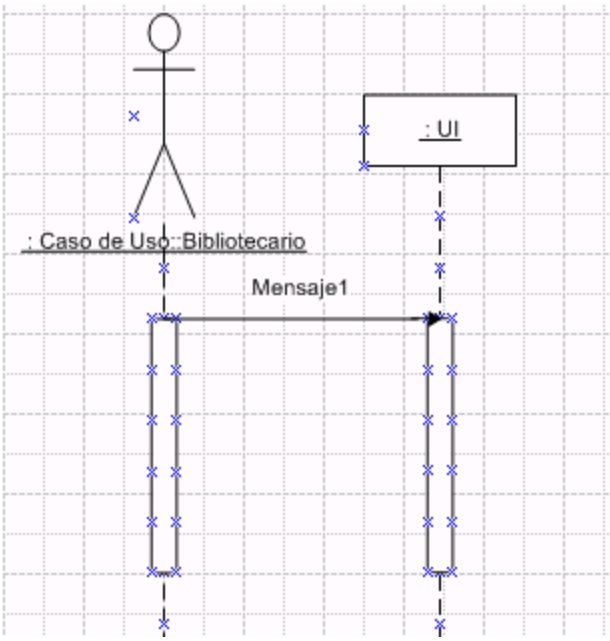


Figura P6.9

Por último vamos a ver como se especifica el mensaje que se va a enviar.

- Da doble click en la forma ***mensaje,*** se desplegarán dos ventanas de propiedades una después de otra, primero la de **propiedades del mensaje** y después la de **propiedades de operación**. La segunda ventana aparece porque la clase UI no tiene ninguna operación (método) del cual elegir.
- Nuestro método se llamará ***IntroduceClaveDePrestatario.***
- Escribe el nombre del método en la ventana propiedades de operación (*Figura P6.10*) y presiona aceptar dos veces. Esto agregará el método automáticamente a la clase UI.

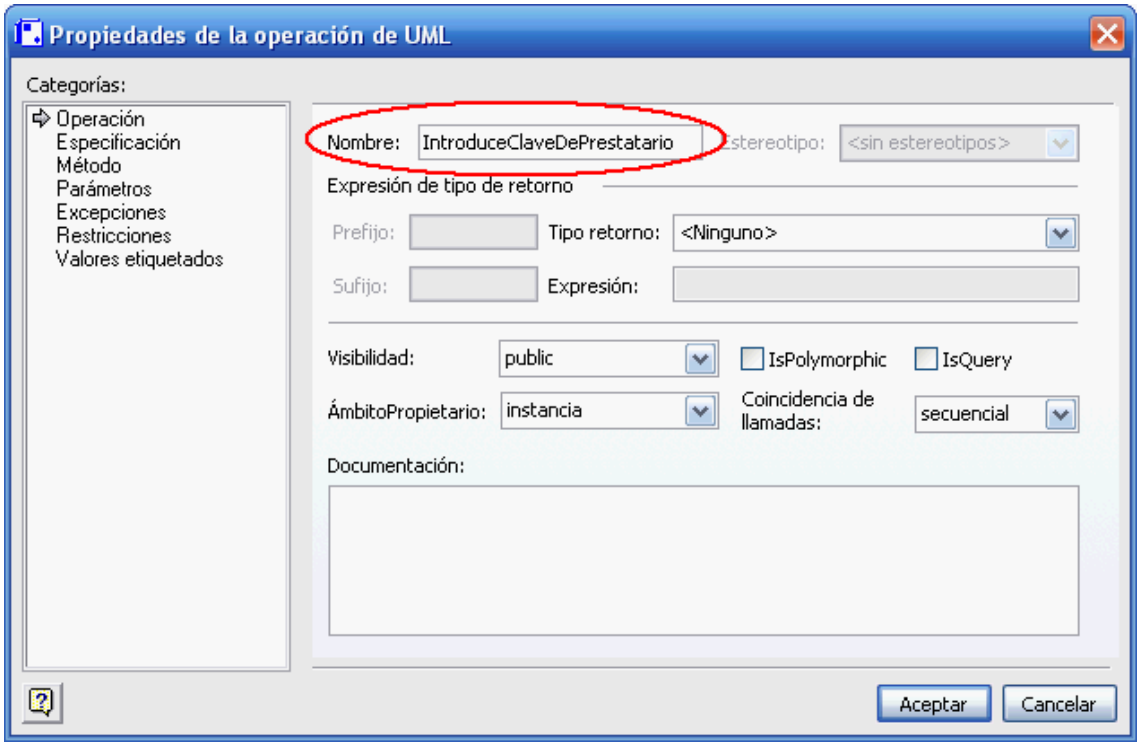


Figura P6.10

- El resultado de nuestra especificación de mensaje debe ser similar a la *Figura P6.11*.

Todas las formas de la plantilla ***secuencia de UML***, se comportan de forma similar a las que vimos en el transcurso de esta práctica. Así que lo único que falta es ver cómo queda nuestro diagrama de secuencia completo para el caso de uso de nuestro ejemplo (*Figura P6.12*):

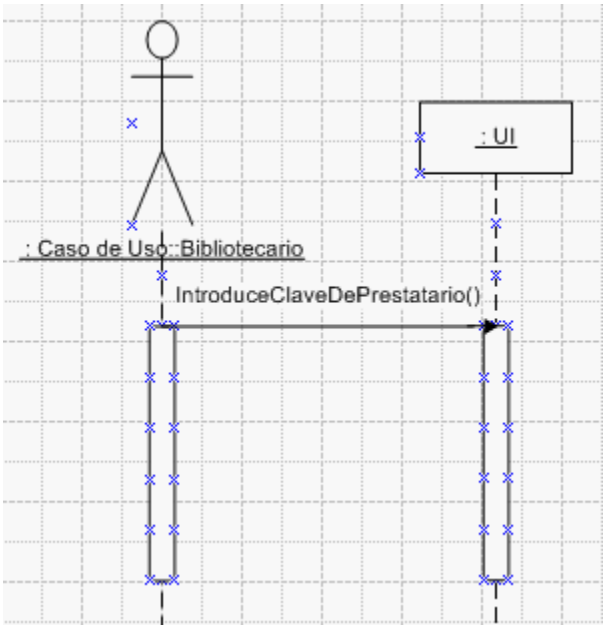


Figura P6.11

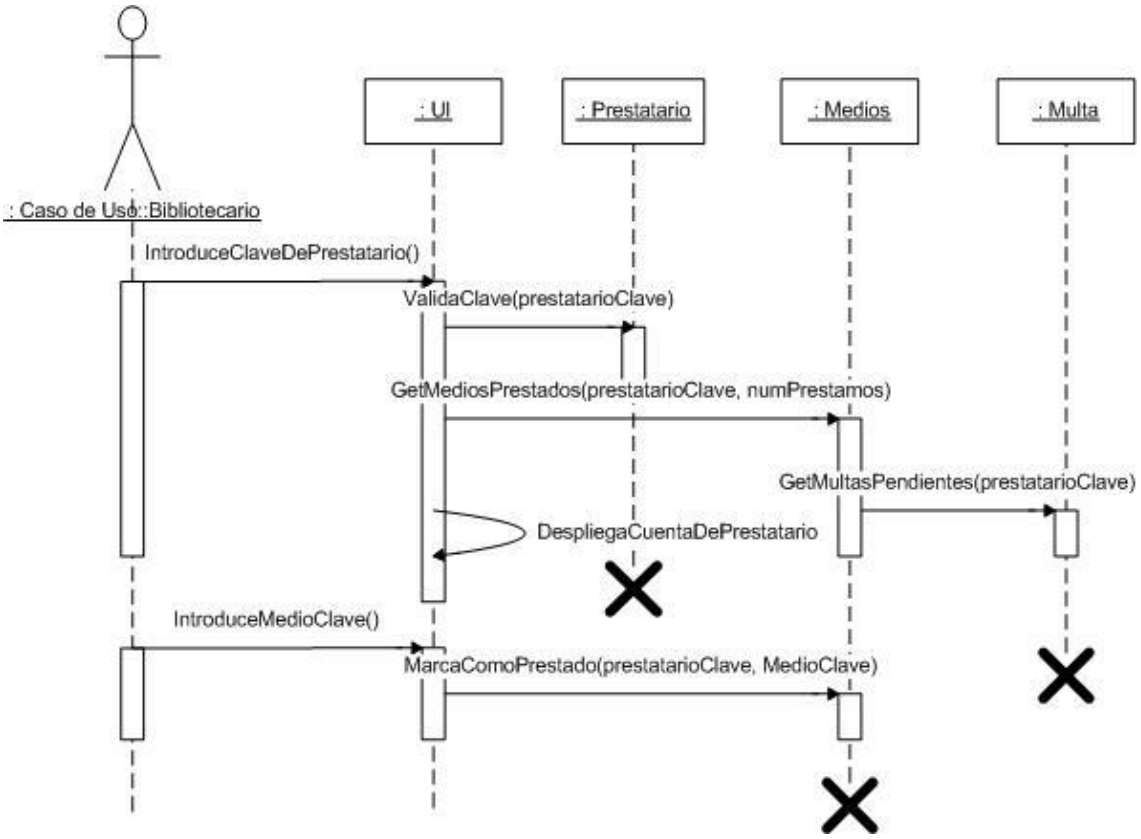


Figura P6.12

Actividades

- Realiza los bosquejos de los diagramas de secuencia para los casos de uso más importantes que conforman tu propio sistema de software.
- Dibuja los diagramas de secuencia con el software Visio.

Cuestionario

- ¿Para qué sirven los diagramas de secuencia?
- ¿Cuáles son los elementos de los diagramas de secuencia?

6.7 Práctica 7 - Diagrama de Estados

Objetivos

- Conocer qué es un diagrama de estados en UML.
- Aprender a diseñar diagramas de estados con ayuda de Microsoft Visio.

Introducción

Diagramas de Estado

Estos diagramas muestran el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación junto con los eventos que permiten pasar de un estado a otro.

Elementos de los diagramas de estados:

Estado: Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto está esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos. Se representa mediante un rectángulo con los bordes redondeados que puede tener tres compartimientos: uno para el nombre, otro para el valor característico de los atributos del objeto en ese estado y otro para las acciones que se realizan al entrar, salir o estar en un estado (entry, exit o do, respectivamente). Existen dos estados especiales el inicial y el final (*Figura P7.1*).

Figura P7.1

Evento: Es una ocurrencia que puede causar la transición de un estado a otro de un objeto. Esta ocurrencia puede ser una de varias cosas:

- Condición que toma el valor de verdadero o falso
- Recepción de una señal de otro objeto en el modelo
- Recepción de un mensaje
- Paso de cierto período de tiempo después de entrar al estado

Envío de mensajes: Además de mostrar la transición de estados por medio de eventos, puede representarse el momento en el cual se envían los mensajes a otros objetos.

Transición simple: Una transición simple es una relación entre dos estados que indica que un objeto en el primer estado puede entrar al segundo estado y ejecutar ciertas operaciones (*Figura P7.2*), cuando un evento ocurre y si ciertas condiciones son satisfechas. Se representa como una línea sólida entre dos estados, que puede venir acompañada de un texto con el siguiente formato:

Evento "[" condición"]"/" acción "^"clausula

Evento es la descripción del evento que da lugar la transición. **Condición** es la condición adicional al evento, necesaria para que la transición ocurra. La **acción** es un mensaje al objeto que se ejecuta como resultado de la transición y el cambio de estado y **cláusula** son las acciones adicionales que se ejecutan con el cambio de estado, por ejemplo, el envío de eventos a otras clases.

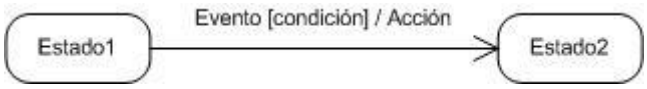


Figura P7.2

Acción: Podemos especificar la solicitud de un servicio a otro objeto como consecuencia de la transición. Se puede especificar el ejecutar una acción como consecuencia de entrar, salir, estar en un estado, o por la ocurrencia de un evento.

Ejemplo

Para ejemplificar el diagrama de estados retomaremos el ejemplo de nuestro *sistema de software para biblioteca.* El diagrama que se presenta en la *Figura P7.3* muestra los estados por los que se atraviesa cuando se hace una solicitud de préstamo.

Desarrollo

A continuación se muestra como crear una forma ***estado***:

- Arrastra una forma ***estado*** desde la plantilla ***Gráfico de estados UML*** hasta el área de dibujo.
- Da doble click sobre la forma y en la ventana de propiedades (*Figura P7.4*) escribe el nombre que tendrá el estado.

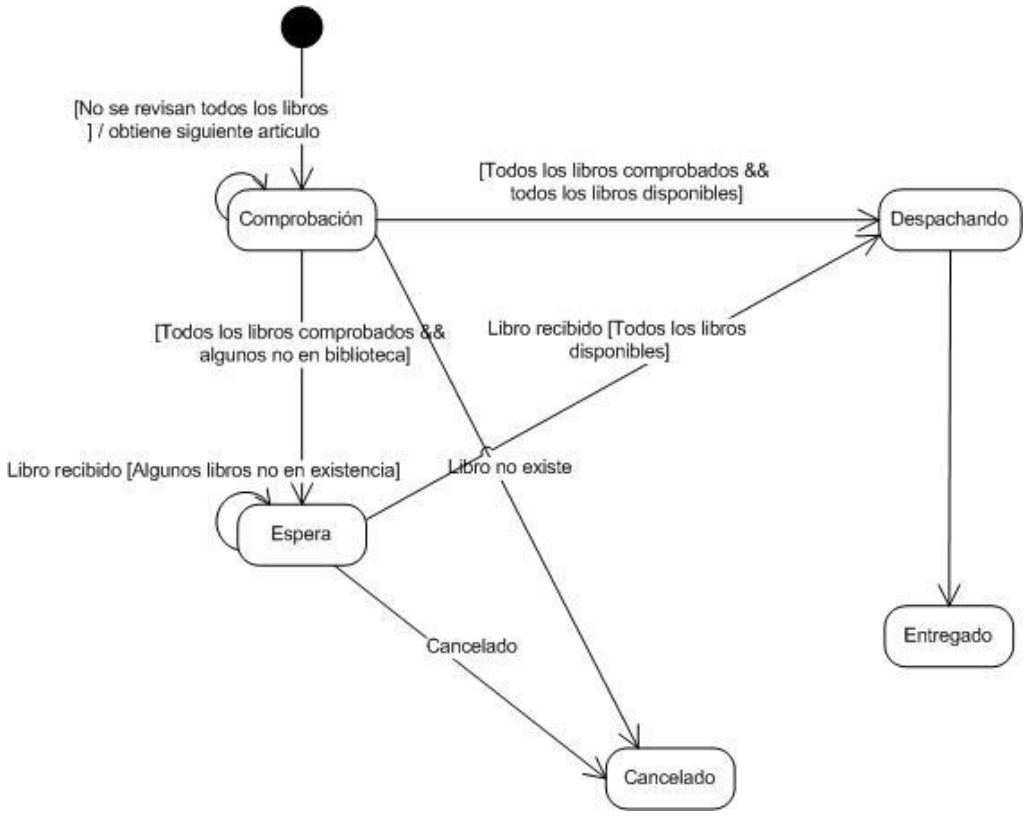


Figura P7.3. Solicitud de préstamo

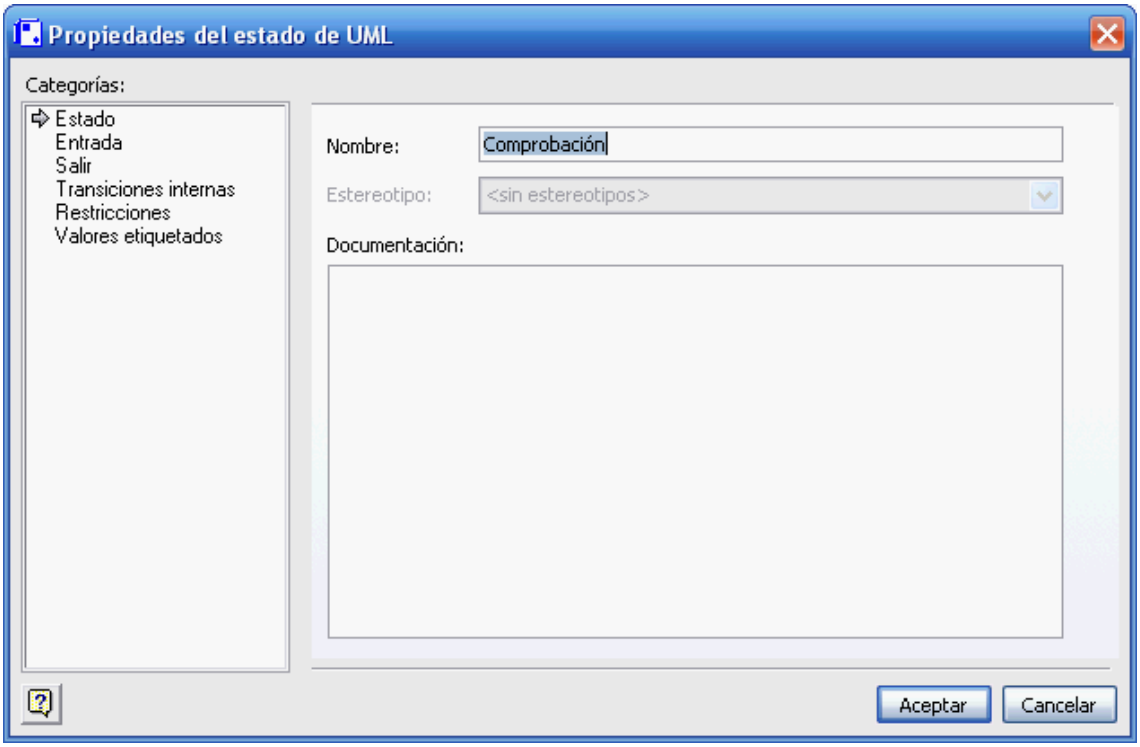


Figura P7.4

El nombre del estado se mostrará dentro de la forma (*Figura P7.5*).

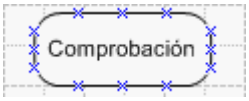


Figura P7.5

Ahora mostraremos como agregar acciones, eventos y condiciones a las flechas que representan una transacción.

- Crea una ***transacción*** (la forma de flecha en la plantilla grafico de estados UML) en el área de dibujo y entra en sus propiedades. Selecciona la categoría acciones y crea una nueva acción como lo hiciste en los pasos anteriores. Elije el tipo de acción que más se adecúe a tus necesidades y presiona aceptar (*Figura P7.6*).

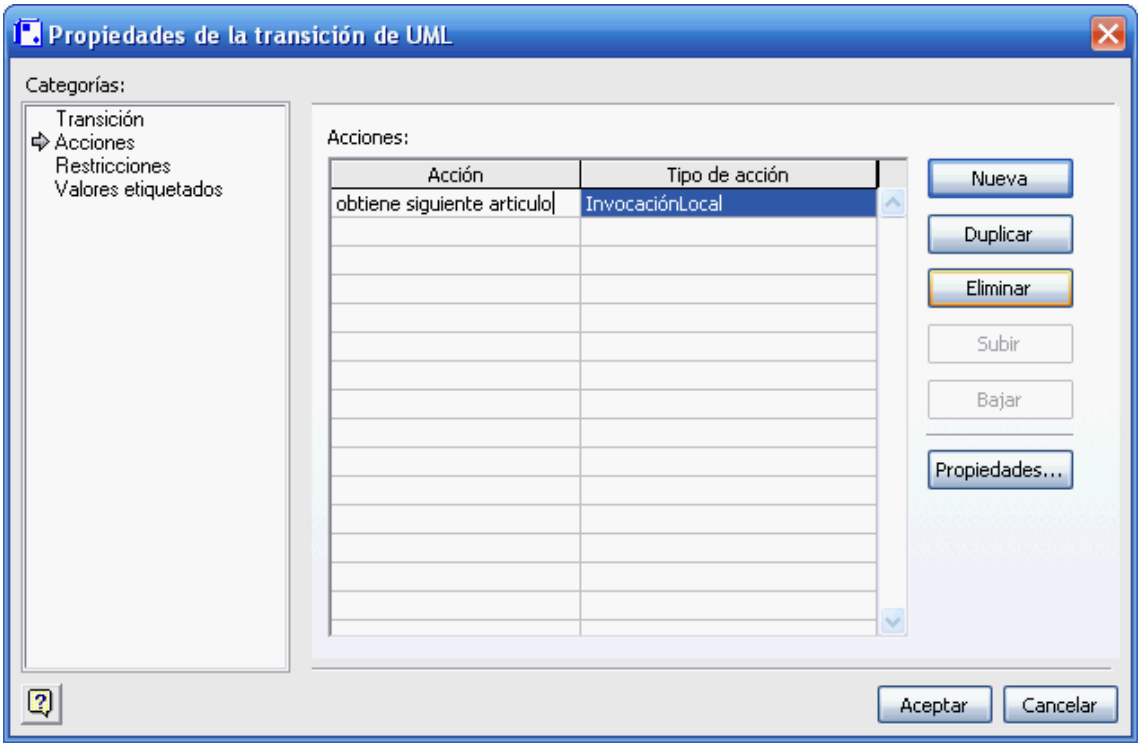


Figura P7.6

- Para agregar una condición a la transición: Abre la ventana de propiedades de la transición (si no está abierta) y marca la opción Protección. Elije ***Texto*** en el combo box lenguaje y el área de texto ***cuerpo*** escribe el texto de tu condición (*Figura P7.7*).

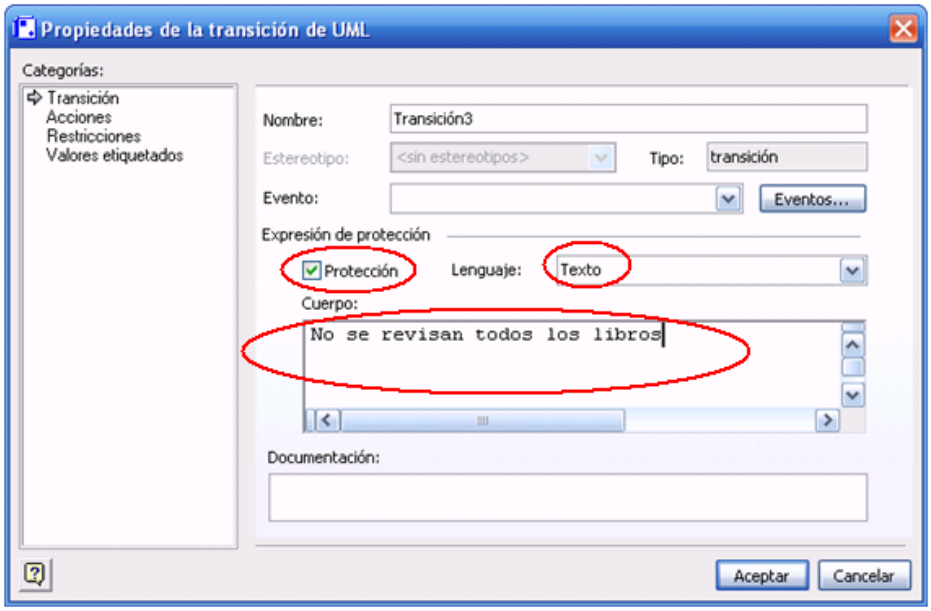


Figura P7.7

- Ahora tu transacción (la flecha) muestra la condición y la acción que realiza (*Figura P8.8*).

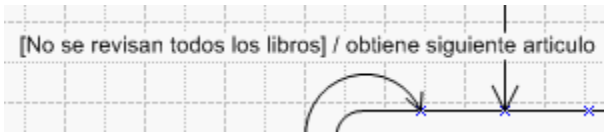


Figura P7.8

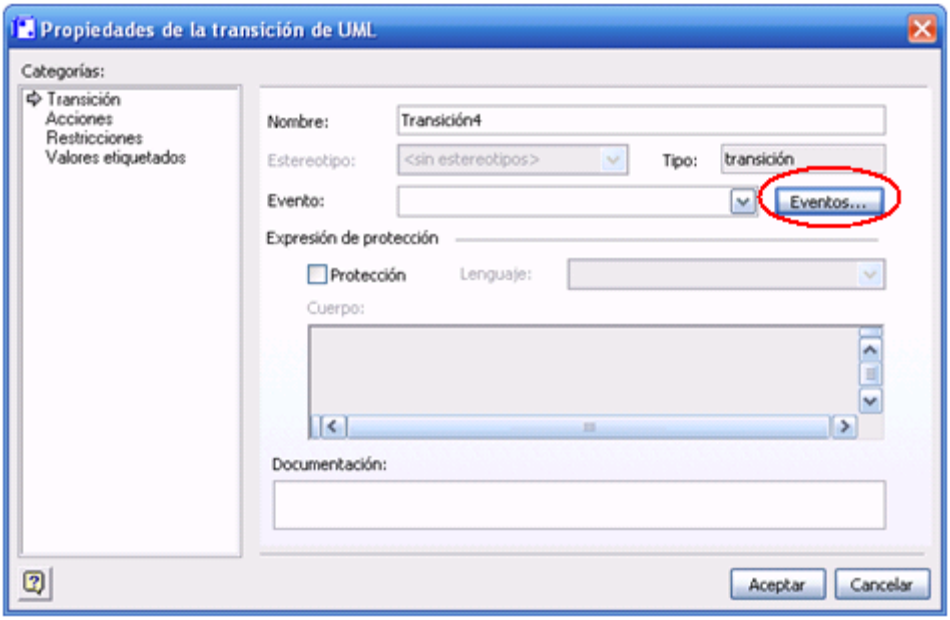


Figura P7.9

- Para agregar un evento a una transacción: Abre la ventana de propiedades de la transición (si no está abierta) y presiona el botón eventos (*Figura P7.9*).

- Aparecerá la ventana de eventos. Selecciona ***tipos de datos C#*** y presiona el botón ***Nuevo***. Se te solicitará que selecciones el tipo de evento y luego se desplegará otra ventana en la que podrás escribir el nombre que quieres que tenga tu evento, presiona aceptar para volver a la ventana de eventos (*Figura P7.10*) y presiona aceptar una vez más para volver a la ventana de propiedades.

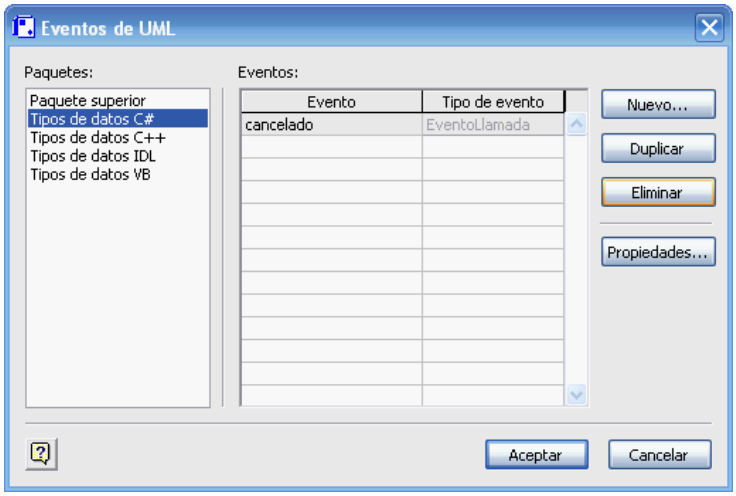


Figura P7.10

- Una vez que estás en la ventana de propiedades sólo falta que selecciones el evento que quieres que tu transacción muestre (*Figura P7.11*).

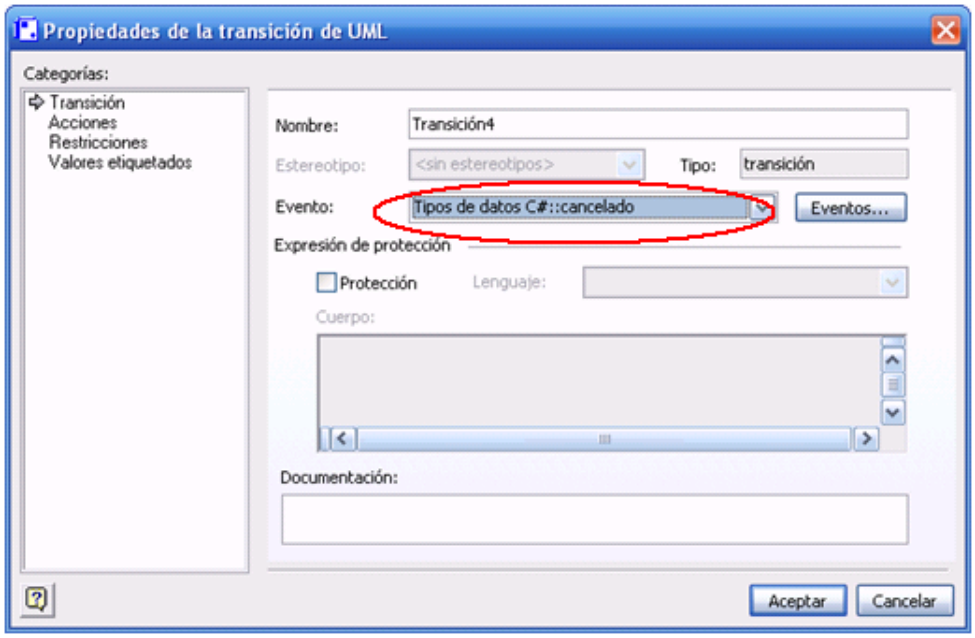


Figura P7.11

- Tu transacción ya debe mostrar el evento que provoca el cambio de estado en el diagrama (*Figura P7.12*).

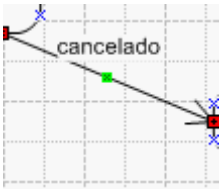


Figura P7.12

Esto cubre de manera general, el uso de las formas de la plantilla *gráficos de estado UML*. Ahora ya tienes las habilidades necesarias para crear tu propio diagrama de estados.

Actividades

- Realizar los bosquejos de los diagramas de estados más significativos de tu sistema de software.
- Dibujar los diagramas de estados con el software Visio.

Cuestionario

- ¿Para qué sirven los diagramas de estados?
- ¿Cuáles son los elementos de los diagramas de estados?

6.8 Práctica 8 - Diseño de la arquitectura con diagramas de paquetes.

Objetivos

- Aprender a realizar el diseño de la arquitectura usando el diagrama de paquetes.
- Aprender a dibujar un diagrama de paquetes con ayuda de Microsoft Visio.

Introducción

En la fase de diseño se determina la arquitectura general del sistema que mejor satisface los requerimientos, también se define cómo debe realizar su función el sistema. El resultado obtenido de la etapa de diseño facilita enormemente la implementación del sistema, pues proporciona la estructura básica del mismo y cómo los diferentes componentes actúan y se relacionan entre ellos.

La elección de la arquitectura del sistema es un punto básico de la fase de diseño así como del proyecto en sí. En esta práctica aprenderemos a usar la arquitectura en capas. Esta arquitectura es descrita por medio de los diagramas de paquete modelados con UML, poniendo un paquete por cada capa.

Arquitectura en tres capas

Una capa es una abstracción que toma el resultado de la capa inferior, efectúa su función y entrega su resultado a la capa superior. El modelo de tres capas tiene, como su nombre lo dice, tres capas que son:

- **Capa de Presentación o interfaz (interfaz con el usuario):** Aquí se ponen los elementos que interactuarán directamente con el usuario, que forman la interfaz de usuario. Esta capa es responsable de presentar información y de interactuar con la capa media.
- **Capa Lógica de la aplicación (reglas del negocio):** Aquí se definen los elementos que implementan la lógica de la aplicación, es decir, se definen las funcionalidades necesarias.
- **Capa de Almacenamiento (bases de datos):** En esta capa se encuentran los elementos que guardan la información, los elementos persistentes.

Diagramas de paquetes

La arquitectura del sistema será representada con diagramas de paquetes como se explica a continuación.

Diagramas de paquete: Estos diagramas son otra herramienta de UML. Muestran como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los

diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. Los **paquetes** están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Los paquetes pueden incluir clases, interfaces, otros paquetes, código HTML, etc. A cada paquete se le da un nombre y éste puede representar una funcionalidad o un caso de uso.
Usando la arquitectura de tres capas, tenemos tres paquetes, uno para cada capa:

- **Paquete de Interfaz de usuario**: Este paquete puede tener uno o varios paquetes que contengan la interfaz de usuario y contiene los elementos de la capa de interfaz.

- **Paquete de lógica de la aplicación:** Este paquete contiene varios paquetes, uno para cada caso de uso del sistema, otro paquete para los servicios que necesitan todos estos paquetes como es la conexión a la base de datos. Este paquete es responsable de implementar las operaciones solicitadas por los clientes.

- **Paquete de almacenamiento**. Este paquete puede contener varios paquetes que contienen la base de datos. Es el paquete responsable de la manipulación de datos.

Propósito de los diagramas de paquetes: Dividir un sistema en varios subsistemas. Se usan para simplificar el modelo agrupando los elementos relacionados.

Elementos de Diagrama de Paquetes

- **Paquete:** Representa un grupo de elementos (casos de uso, clases, componentes, otros paquetes) relacionados según algún criterio. Se representa por medio de un fólder al cual se le pone el nombre del paquete (*Figura P8.1*).

Figura P8.1

- **Asociación:** Establece una relación entre dos paquetes que requieren de los servicios uno del otro. Esta relación es bidireccional (*Figura P8.2*).

Figura P8.2

- **Dependencia entre paquetes:** Dos paquetes son dependientes si por lo menos un elemento de un paquete depende de algún elemento de otro paquete, es decir, si un cambio en el primer paquete afecta al paquete que depende de él. Por ejemplo, cuando

una clase de un paquete llama a una clase del otro ó la importación de un paquete. Se representa por medio de una línea discontinua con flecha (*Figura P8.3*).

Figura P8.3

- **Generalización entre paquetes:** Con los paquetes se puede aplicar generalización, esto significa que el paquete específico debe conformarse a la interfaz del paquete general (*Figura P8.4*).

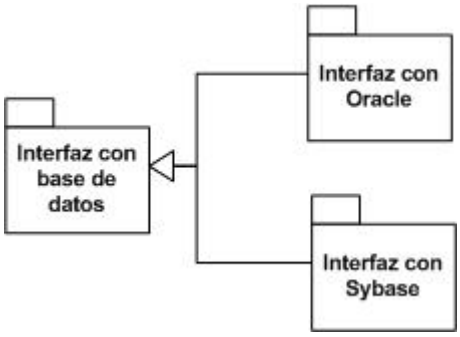


Figura P8.4

- **Realización**: Es un contrato que promete que un paquete será implementado por otro (*Figura P8.5*).

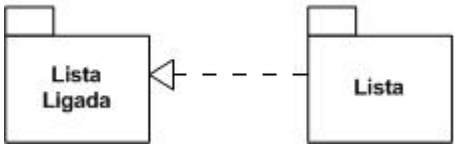


Figura P8.5

Ejemplo

Este es el ejemplo de un diagrama de paquetes en el que se implementa la arquitectura de tres capas (*Figura P8.6*).

- La primera capa representa los paquetes que intervienen en la capa de presentación o de interfaz.
- La segunda capa representa los paquetes que intervienen en la capa de lógica del sistema.
- La tercera capa representa los paquetes que intervienen en la capa del almacenamiento de datos.

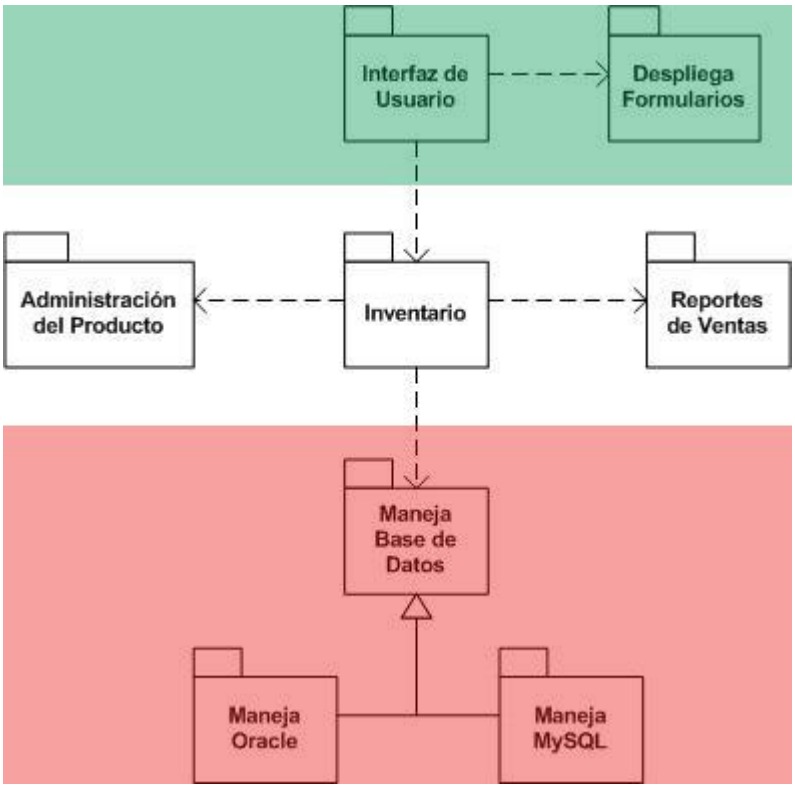


Figura P8.6

Desarrollo

El diagrama de paquetes se realiza utilizando la misma plantilla (stencil) que utilizamos para el diagrama de clases (la plantilla e*structura estática de UML*). Por lo tanto ya debes poseer las habilidades necesarias para realizar este diagrama.

Actividades

- Diseñar el diagrama de paquetes para tu sistema de software. El diagrama deberá estar aplicado a la arquitectura en tres capas que se explicó en esta práctica.
- Dibujar el diagrama de paquetes con el software Visio.

Cuestionario

- ¿Para qué sirven los diagramas de paquetes?

- ¿Cuáles son los elementos de los diagramas de paquetes?

6.9 Práctica 9 - Diagrama de Distribución

Objetivos

- Conocer que es un diagrama de distribución en UML.
- Aprender a dibujar un diagrama de distribución con ayuda de Microsoft Visio.

Introducción

Los diagramas de distribución muestran la disposición física de los distintos **nodos** que componen un sistema y el reparto de los componentes sobre dichos nodos.

Un **nodo** es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento.

Los nodos se utilizan para modelar la **topología del hardware** sobre el que se ejecuta el sistema. Representa típicamente un procesador o un dispositivo sobre el que se pueden desplegar los componentes (*Figura P9.1*).

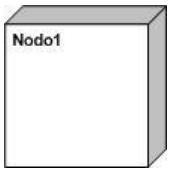


Figura P9.1

Los **componentes** son los elementos que participan en la ejecución de un sistema. Los nodos son los elementos donde se ejecutan los componentes.

Los **componentes** representan el empaquetamiento físico de los elementos lógicos. Los **nodos** representan el despliegue físico de los componentes.

La **relación** entre un nodo y el componente que despliega puede mostrarse con:

- Una relación de dependencia (*Figura P9.2*).

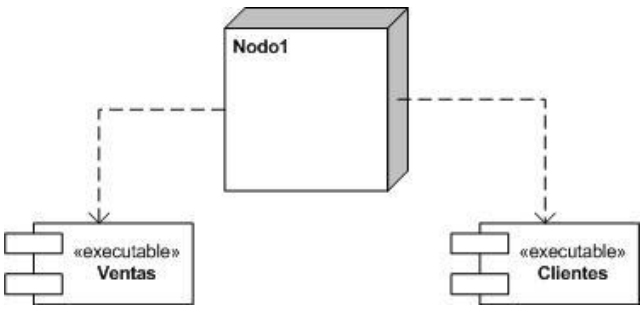


Figura P9.2

- Listando los nodos desplegados en un compartimiento adicional dentro del nodo (*Figura P9.3*).

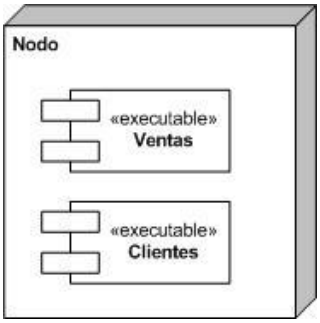


Figura P9.3

Los estereotipos permiten precisar la naturaleza del equipo:

- **Procesadores:** Nodo con capacidad de procesamiento. Puede ejecutar un componente.
- **Dispositivos:** Nodo sin capacidad de procesamiento. Representa cualquier otro dispositivo hardware.

Los nodos se relacionan mediante conexiones bidireccionales (en principio) que pueden a su vez estereotiparse. Las conexiones se modelan como asociaciones, con todas las características que implica (*Figura P9.4*).

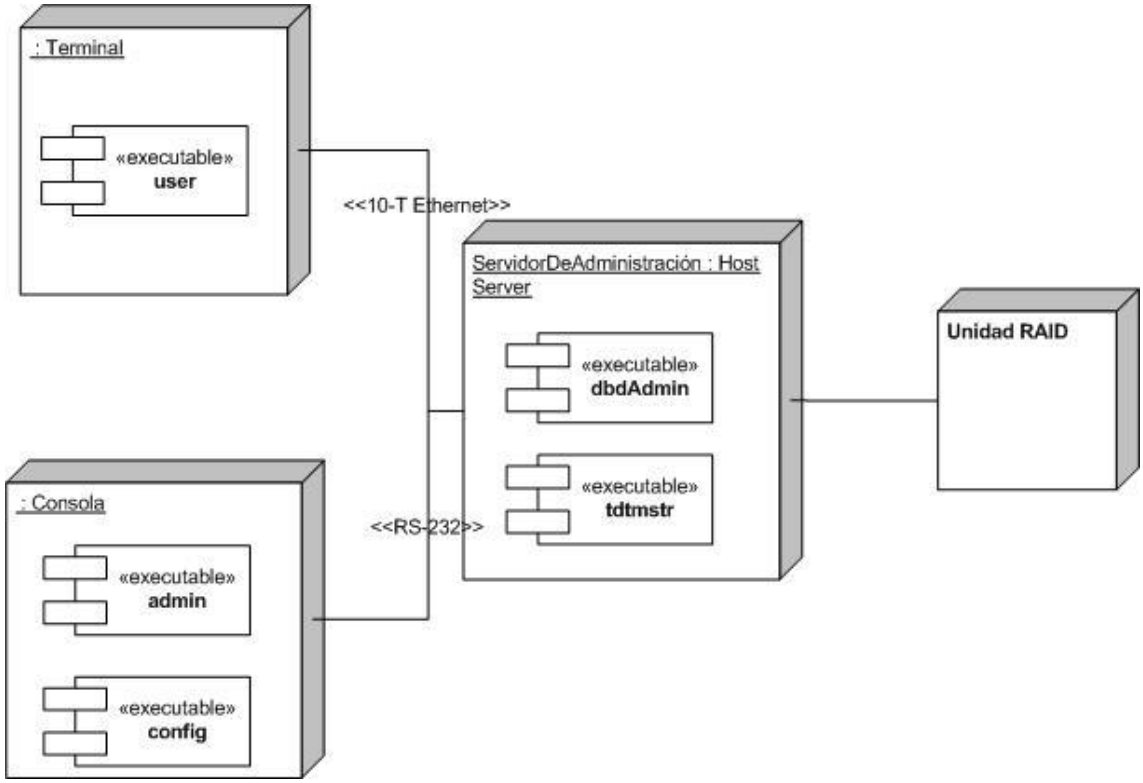


Figura P9.4 Ejemplo

Desarrollo

En el menú Archivo, selecciona Nuevo, Software y, a continuación, haz click en Diagrama de modelo de UML.

En la vista de árbol, haz click con el botón secundario en el paquete o subsistema en el que deseas incluir el diagrama de implementación, selecciona Nuevo y haz click en Diagrama de implementación.

Aparece una página en blanco y la galería de símbolos *Implementación de UML* pasa a estar activa. El área de trabajo muestra el texto 'Implementación' como marca de agua y se agrega un icono a la vista de árbol para representar el diagrama.

Arrastra una forma **Nodo** a la página de dibujo. Arrastra la forma **Componente** al nodo. Arrastra un asa de selección del nodo para cambiar su tamaño. Agrega tantos nodos y componentes como tenga tu diagrama.

Para incluir elementos en un nodo de un diagrama de implementación:

- Arrastra una forma **Nodo** o **Instancia de nodo** de la galería de símbolos Implementación de UML a la página de dibujo.

- Haz doble click en el **nodo** para agregar un nombre, atributos, operaciones y otros valores de propiedades.

- Arrastra un asa de selección de esquina para cambiar el tamaño del nodo y poder albergar los elementos que va a contener.

- Arrastra las formas **Componente** o **Instancia de componente** sobre la forma **Nodo** y, a continuación, conecta las formas con relaciones de dependencia donde corresponda.

- Haz doble click en cada nodo. En el cuadro de diálogo **Propiedades del nodo de UML** (*Figura P9.5*), haz click en **Componentes**. En la sección **Elija los componentes que implementa este nodo**, selecciona los apropiados y, a continuación, haz click en Aceptar.

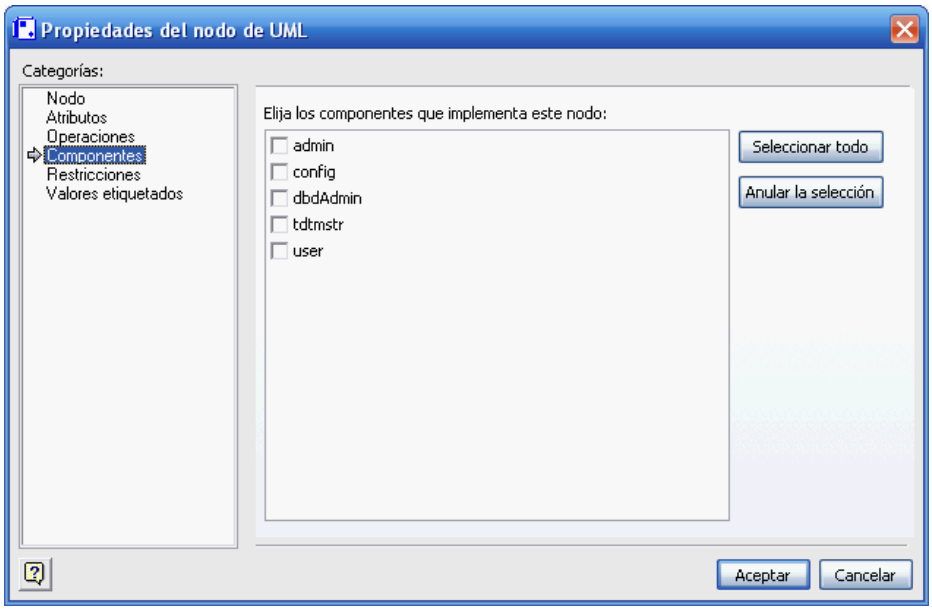


Figura P9.5.

- Haz doble click en cada componente. En el cuadro de diálogo **Propiedades del componente de UML**, haz click en **Nodos**. En la sección **Elija los nodos que implementan este componente**, selecciona los nodos apropiados y haz click en Aceptar (*Figura P9.6*).

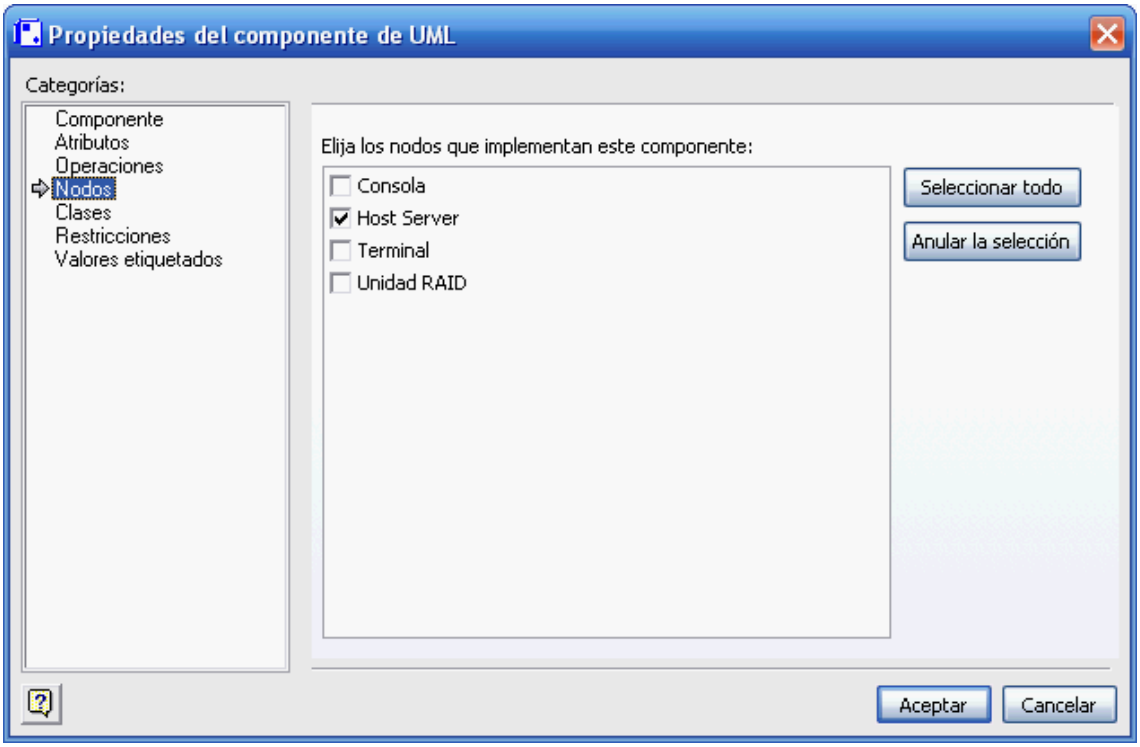


Figura P9.6.

Utiliza las formas comunicados para indicar la relación existente entre los nodos (como en prácticas anteriores).

Utiliza las formas dependencia para indicar las relaciones existentes entre componentes y objetos (como en prácticas anteriores), así como entre componentes e interfaces de otros componentes.

Actividades

- Realizar el bosquejo del diagrama de distribución que tendrá tu propio sistema de software.
- Dibujar el diagrama de distribución con el software Visio.

Cuestionario

- ¿Para qué sirven los diagramas de distribución?
- ¿Cuáles son los elementos de los diagramas de distribución?

6.10 Práctica 10 - Programación en C#

Objetivos

- Aprender la sintaxis del lenguaje **C#**.
- Conocer cómo se maneja la programación orientada a objetos en **C#**.
- Aprender a crear clases, métodos y objetos en **C#**.
- Aprender a declarar interfaces y clases abstractas.

Introducción

Lenguaje C#

C# es un lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Aunque es posible escribir código en muchos otros lenguajes para esta plataforma, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes, ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET.

La sintaxis y estructuración de C# son muy similares a las de C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL (Base Class Library) usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el .NET Framework SDK.

Aspectos Léxicos

Comentarios: Los comentarios se realizan con (**//**) en caso de que el comentario sea de una sola línea y (**/* */**) para abarcar varias líneas.

Operadores aritméticos: Los operadores aritméticos incluidos en C# son los típicos de suma (**+**), resta (**-**), producto (*****), división (**/**) y módulo (**%**).

Operadores lógicos: Los operadores que permiten realizar las operaciones lógicas típicas son “and” (**&&**), “or” (**||**), “not” (**!**) y “xor” (**^**).

Operadores relacionales: Los operadores relacionales son los típicos de igualdad (**==**), desigualdad (**!=**), "mayor que" (**>**), "menor que" (**<**), "mayor o igual que" (**>=**) y "menor o igual que" (**<=**).

Operadores de asignación: Para realizar asignaciones en C# se usa el operador =. También se incluyen los operadores **+=**, **-=**, ***=**, **/=**, **%=** los cuales funcionan de la misma forma que los operadores de Java. De igual forma se incluyen los operadores **++** y **--**.

Operador condicional: Es el único operador incluido en C# que toma 3 operandos:

<condición> ? <expresión1> : <expresión2>

Se usa así:

```
b = (a>0)? a : 0;
```

La variable b toma el valor de a si la condición es cierta ó el valor 0 en caso contrario.

Operador de cadenas: La concatenación de cadenas se realiza con el operador +.

Operaciones de obtención de información sobre tipos: De todos los operadores que nos permiten obtener información sobre tipos de datos el más importante es **typeof**, cuya forma de uso es:

typeof(<nombreTipo>)

Este operador devuelve un objeto de tipo System.Type con información sobre el tipo de nombre <nombreTipo> que podremos consultar a través de los miembros ofrecidos por dicho objeto. Esta información incluye detalles tales como cuáles son sus miembros, cuál es su tipo padre o a qué espacio de nombres pertenece.

Si lo que queremos es determinar si una determinada expresión es de un tipo u otro, entonces el operador a usar es **is**, cuya sintaxis es la siguiente:

<expresión> **is** <nombreTipo>

El significado de este operador es el siguiente: se evalúa <expresión>. Si el resultado de ésta es del tipo cuyo nombre se indica en <nombreTipo> se devuelve true; y si no, se devuelve false.

Modificadores

C# soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores public, private y protected, C# añade un cuarto modificador llamado internal, que puede combinarse con protected e indica que al elemento a cuya definición precede sólo puede accederse desde su mismo ensamblado.

Arreglos

Los arreglos unidimensionales se manejan de la misma forma que en Java, por ejemplo:

```
int[] unArreglo = new int[4]; //Define un arreglo unidimensional de enteros.
```

Una arreglo multidimensional se define con una sintaxis similar a la usada para declarar arreglos unidimensionales pero separando las diferentes dimensiones mediante comas (,)

Por ejemplo:

```
int[,] arregloMult = new int[3,4];
```

Define un arreglo multidimensional de elementos de tipo int que conste de 12 elementos distribuidos en una estructura de 3x4.

Condiciones y Ciclos

Las sentencias condicionales de C# tienen la misma sintaxis y se manejan de la misma forma que en Java:

Condición if

```
if (condición){ ... }
else { ... }
```

Condición Switch

```
switch (expresión){
        case <valor1>:<bloque>
                        break;
        case <valor2>:<bloque>
                        break;
}
```

Ciclo while

```
while (condición){
<bloque>
 }
```

Ciclo do...while

```
do{
 <bloque>
 }while (condición);
```

Ciclo for

```
for ( <inicialización> ; <condición> ; <modificación>) {  <bloque>  }
```

El lenguaje C# cuenta con una estructura cíclica especial para recorrer colecciones de elementos, su sintaxis es:

```
foreach (<TipoElemento> <elemento> in <colección>){<bloque>}
```

En este ejemplo se imprimen en pantalla los elementos de un arreglo:

```
int[] unArreglo = new int[4];

foreach (int a in unArreglo){

        Console.WriteLine( a );

}
```

Clases

Esta es la definición típica de una clase en C#:

```
class Persona{

        private string Nombre;   // Atributos de clase

        private int Edad;            // Atributos de clase


        Persona (string Nombre, int Edad){ // Constructor

                this.Nombre = Nombre;

                this.Edad = Edad;

        }


        void Cumpleaños(){ // Método

                Edad++;

        }

}
```

Como notarás es prácticamente igual a la definición de una clase en Java, la única diferencia es que los atributos de clase y los métodos empiezan con una letra mayúscula (por convención de C#).

Es importante señalar que en C# todo, incluido los literales, son objetos que derivan de la clase inicial Object por lo tanto pueden contar con métodos a los que se accedería tal y como se ha explicado. Por ejemplo:

```
string s = 12.ToString();
```

En donde se muestra que el numero 12 puede tomarse como un objeto de la clase Object.

Objetos

Un objeto en C# (instancia de clase) se crea de la misma forma que en Java:

```
Persona p = new Persona("José", 22);
```

En el ejemplo de la definición de clase anterior, puedes notar que al igual que en Java, se utiliza la palabra reservada **this** para hacer referencia al objeto actual.

Herencia

Las clases que derivan de otras se definen usando la siguiente sintaxis:

```
class <nombreHija>:<nombrePadre> {

        <contenidoHija>

}
```

A los atributos y métodos definidos en <contenidoHija> se le añadirán los que hubiésemos definido en la clase padre. Por ejemplo, a partir de la clase Persona puede crearse una clase Trabajador así:

```
class Trabajador: Persona {
        public int Sueldo;


        Trabajador(string nombre, int edad, int sueldo): base(nombre, edad){
        // Inicializamos cada Trabajador en base al constructor de Persona
                Sueldo = sueldo;
        }
}
```

Los objetos de esta clase Trabajador contarán con los mismos atributos y métodos que los objetos Persona y además incorporarán un nuevo campo llamado Sueldo que almacenará el dinero que cada trabajador gane. Nótese además que a la hora de escribir el constructor de esta clase ha sido necesario escribirlo con una sintaxis especial consistente en preceder la llave de apertura del cuerpo del método de una estructura de la forma:

```
: base(<parametrosBase>)
```

A esta estructura se le llama inicializador base y se utiliza para indicar cómo deseamos inicializar los campos heredados de la clase padre. No es más que una llamada al constructor de la misma con los parámetros adecuados, y si no se incluye el compilador consideraría por defecto que vale :base(), lo que sería incorrecto en este ejemplo debido a que Persona carece de constructor sin parámetros.

Para crear una instancia de esta clase se utiliza esta sentencia:

```
Trabajador p = new Trabajador("José", 22,100000);
```

Hay casos en los que deseamos que una clase hija, utilicé un método que se llame igual a un método de la clase padre, pero que esté método realicé acciones diferentes a las del método del padre. Es decir sobrescribir o recargar un método. Podemos realizar esto mediante las palabras reservadas **virtual** y **override**, ejemplo:

```
class Persona{
        public string Nombre;
        public int Edad;

        public virtual void Cumpleaños() {
                Console.WriteLine("Incrementada edad de persona");
        }

        public Persona (string nombre, int edad, string nif){
        // Constructor de Persona
                Nombre = nombre;
                Edad = edad;
        }
}
class Trabajador: Persona {
        public int Sueldo;

        Trabajador(string nombre, int edad, string nif, int sueldo): base(nombre, edad, nif) {
                Sueldo = sueldo;
        }

        public override Cumpleaños()
        {
                Edad++;
                Console.WriteLine("Incrementada edad de persona");
        }
}
```

Como se muestra en el ejemplo, para sobrecargar el método de una clase padre, es necesario preceder el nombre del método en la clase padre por la palabra **virtual** y preceder el nombre del método en la definición de la clase hija por la palabra **override**.

El lenguaje C# impone la restricción de que toda redefinición de método que queramos realizar incorpore la partícula **override** para forzar a que el programador esté seguro de que

verdaderamente lo que quiere hacer es cambiar el significado de un método heredado. Así se evita que por accidente defina un método del que ya exista una definición en una clase padre. Además, C# no permite definir un método como **override** y **virtual** a la vez, ya que ello tendría un significado absurdo: estaríamos dando una redefinición de un método que vamos a definir.

También es importante señalar que para que la redefinición sea válida ha sido necesario añadir la partícula public a la definición del método original (virtual), pues si no se incluyese se consideraría que el método sólo es accesible desde dentro de la clase donde se ha definido,

Para acceder a los métodos de la clase base o super clase, C# proporciona la palabra reservada base (el equivalente de la palabra reservada super en Java). Ejemplo:

```
public override void Cumpleaños(){

        base.Cumpleaños();

}
```

Clases Abstractas

Para definir una clase abstracta se antepone **abstract** a su definición, como se muestra en el siguiente ejemplo:

```
public abstract class A{
        public abstract void F();
}

abstract public class B: A{
        public void G() {}
}

class C: B {
        public override void F() {}
}
```

Las clases A y B del ejemplo son abstractas, y como puede verse es posible combinar en cualquier orden el modificador **abstract** con modificadores de acceso.

La utilidad de las clases abstractas es que pueden contener métodos para los que no se dé directamente una implementación sino que se deja en manos de sus clases hijas darla. No es obligatorio que las clases abstractas contengan métodos de este tipo, pero sí la clase contiene un método abstracto como abstracta a toda la que tenga alguno. Estos métodos se definen precediendo su definición del modificador **abstract** y sustituyendo su código por un punto y coma (;), como se muestra en el método F() de la clase A del ejemplo (nótese que B también ha de definirse como abstracta porque tampoco implementa el método F() que hereda de A)

Véase que todo método definido como abstracto es implícitamente virtual, pues si no sería imposible redefinirlo para darle una implementación en las clases hijas de la clase abstracta donde esté definido. Por ello es necesario incluir el modificador **override** a la hora de darle implementación como se muestra en el método F de la clase C. Es redundante marcar un método como **abstract** y **virtual** a la vez (de hecho, hacerlo provoca un error al compilar)

Clases y métodos sellados

Una clase sellada es una clase que no puede tener clases hijas, y para definirla basta anteponer el modificador **sealed** a la definición de una clase normal. Por ejemplo:

```
sealed class ClaseSellada {

}
```

Aparte de para sellar clases, también se puede usar **sealed** como modificador en la redefinición de un método para conseguir que la nueva versión del mismo que se defina deje de ser virtual y se le puedan aplicar las optimizaciones arriba comentadas. Un ejemplo de esto es el siguiente:

```
class A {
        public abstract F();
}
class B:A {
        public sealed override F() {}          // F() deja de ser redefinible
}
```

Interfaces

La manera básica de definir una interfaz en C# es muy similar a Java:

```
interface Nombre{

        void Metodo1(int x);

        void Metodo2();

}
```

Al igual que en Java, la definición de una interfaz puede ser mucho más compleja, pero ese tema esta fuera del alcance de esta práctica.

Para implementar una interfaz en una clase, se utiliza una sintaxis muy similar a la que utilizamos para extender una clase.

```
interface InterA{
    void F();
}

class C1: InterA{
    public void F(){
            Console.WriteLine("El F() de C1");
    }
}
```

En esta práctica se han explicado de forma general las principales características del lenguaje de programación C#, para mayor referencia consulta los libros y tutoriales de la bibliografía.

Actividades

- De acuerdo a tu diagrama de clases, codifica las definiciones de todas las clases que conforman tu sistema, eso incluye sus atributos y las firmas de sus métodos.

- De acuerdo al prototipo de interfaz que realizaste en la práctica 4, implementa las *clases parciales* que se encargarán de leer los datos de las pantallas que contienen formularios.

Cuestionario

- ¿Qué diferencias encuentras entre la sintaxis de Java y la de C#?

- ¿Te parece adecuada la forma en la que se maneja la programación orientada a objetos en **C#**?

6.11 Práctica 11 - Características especiales de una aplicación Web en ASP.NET

Objetivos

- Aprender a utilizar sesiones de usuario en una aplicación web de ASP.NET.
- Conocer para que sirven las ***cookies*** y como utilizarlas en ASP.NET.

Introducción

El protocolo de HTTP sobre el cual funcionan las aplicaciones Web está definido de forma que nunca se almacena información acerca de las conexiones y desconexiones que haya habido entre cliente y servidor, es un protocolo ***stateless***. Por lo tanto, cuando necesitamos de algún dato o información de relevancia (preferencias de usuario, número de accesos, recursos visitados…) esté disponible entre diferentes conexiones, es necesario implementar un mecanismo que nos permita almacenar y acceder a dicha información. Para llevar a cabo dicha facilidad, ASP.NET ofrece varias soluciones.

Estado de la Aplicación

Es posible almacenar variables y objetos para utilizarlos a través de la aplicación completa en un objeto especial llamado **Application**. El conjunto de datos almacenados en este objeto se llama **estado de la aplicación**. El objeto **Application** provee métodos que te permiten compartir los datos del estado de la aplicación entre todas las páginas que conforman una aplicación ASP.NET de una forma muy sencilla.

En ASP.NET los estados de aplicación y de sesión están implementados como colecciones (conjunto de parejas nombre-valor). Puedes asignar un valor a una variable de aplicación así:

```
Application["Contador"] = 1;
```

Una vez que la variable existe, cualquier página puede recuperar su valor así:

```
variable = Application["Contador"];
```

Puedes remover un objeto del estado de la aplicación usando el método ***Remove***, así:

```
Application.Remove("Contador");
```

Para remover todos los objetos y variables

```
Application.RemoveAll();
```

Es importante ser cauteloso trabajando con el objeto Application, ya que las variables que han sido agregadas permanecerán allí hasta que sean removidas utilizando alguno de los métodos anteriores. Sí se agregan demasiadas variables y objetos al estado de la aplicación y no se remueven se puede reducir el desempeño de la aplicación.

Podría darse el caso de que dos usuarios entren a la página al mismo tiempo, en este caso la variable **Contador** solo registraría un cambio en su estado. Para evitar estos casos el objeto Application cuenta con dos métodos:

```
Application.Lock();
<Bloque>
Application.UnLock();
```

Un bloque que está encerrado entre estos métodos solo podrá ser accedido por un usuario a la vez.

Sesión de Usuario

Al igual que el estado de la aplicación, el estado de la sesión es una manera importante de almacenar información temporal a lo largo de las páginas de nuestra aplicación. La diferencia es que el estado de la aplicación puede accederse por todos los usuarios, en cambio el estado de la sesión está asociada a un usuario particular que visita la aplicación Web.

Almacenado en el servidor, el estado de la sesión reserva espacio libre en memoria para almacenar las variables y objetos de cada usuario.

El proceso de lectura y escritura de datos en el estado de la sesión es muy similar al proceso de lectura y escritura en el estado de la aplicación. En lugar de utilizar el objeto **Application** usaremos el objeto **Session**. A diferencia del objeto **Application** el objeto **Session** no necesita tener los métodos **lock** y **unlock**.

Al igual que los objetos almacenados en el estado de la aplicación, los objetos del estado de la sesión permanecen en el servidor incluso cuando el usuario abandona la aplicación Web. A diferencias de las variables de aplicación las variables de sesión desaparecen después de un periodo de tiempo en el que el usuario está inactivo. Debido a que los *Web Browsers* no avisan cuando un usuario abandona el sitio Web. ASP.NET solo puede asumir que el usuario ha dejado la aplicación, después de un periodo de tiempo en el cual no recibe peticiones del usuario. Una sesión de usuario expirará después de 20 minutos de inactividad. Esto puede cambiarse incrementando o decrementando la propiedad Timeout del objeto Session.

```
Session.Timeout = 1560; // 15 minutos
```

Objeto Cache

ASP.NET ofrece otro objeto para almacenar datos de una aplicación temporalmente. Al igual que el objeto Application el objeto Cache es una colección y se accede a su contenido de la misma forma que con el objeto Application. Ejemplo: `Cache["variable"] = 21;`

Otra similitud es que ambos objetos son visibles a lo largo de toda la aplicación entre todos los usuarios que han accedido al sitio Web. La diferencia es que los objetos permanecen en el cache hasta que son removidos o hasta que los recursos del servidor disminuyen, en este caso los objetos son removidos del cache en el mismo orden en el que se agregaron.

El objeto cache permite controlar el tiempo de expiración, por ejemplo si queremos agregar una variable al Cache por un periodo de tiempo específico podemos usar el método Insert.

```
Cache.Insert("variable", valor, null,DateTime.MaxValue,
TimeSpan.FromMinutes(10));
```

Cookies

Si quieres almacenar datos que se refieren a un usuario en particular, puedes usar el objeto **Session**, pero existe una desventaja al utilizar esta solución: El contenido del objeto Session se pierde cuando el usuario cierra la ventana del *browser*.

Para almacenar datos durante un periodo largo de tiempo, necesitas usar **cookies**. Las **cookies** son piezas de datos que tu aplicación ASP.NET puede salvar en el browser del usuario, para ser leídas después por la aplicación. Los datos de las cookies no se pierden cuando la ventana del browser se cierra (a menos que el usuario las borre), ya que se almacenan en archivos de texto dentro de la computadora del usuario.

En ASP.NET, una cookie esta representada por la clase **HttpCookie**. Leemos las cookies de un usuario mediante la propiedad **Cookies** del objeto **Request**, y escribimos datos en las cookies de un usuario mediante la propiedad **Cookies** del objeto **Response**. Las cookies expiran por default cuando la ventana del browser se cierra, pero el tiempo de expiración puede establecerse a una fecha determinada, en este caso se convierten en **cookies persistentes**. Considera el siguiente ejemplo:

Código de la página (Default.aspx)

```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="miLabel" runat="server" Text="n"></asp:Label>
    </div>
```

```
</form>
```

Código del archivo code-behind (Default.aspx.cs)

```
protected void Page_Load(object sender, EventArgs e){
// Declaramos la cookie
HttpCookie cookieUsuario;

/* Intentamos obtener el nombre del usuario leyendo la
   cookie NombreUsuario*/
cookieUsuario = Request.Cookies["NombreUsuario"];

if (cookieUsuario == null){
      // Despliga el mensaje
      miLabel.Text = "La Cookie no existe! Creandola ahora";
      // Creando la variable
      cookieUsuario = new HttpCookie("NombreUsuario", "JoeBlack");
      // La cookie expira en un mes
      cookieUsuario.Expires = DateTime.Now.AddMonths(1);
      // Guardando la cookie en el cliente (pc del usuario)
      Response.Cookies.Add(cookieUsuario);
}
else
{
      // Despliega el mensaje
      miLabel.Text = "Bienvenido de nuevo "+cookieUsuario.Value;
}

}
```

El código es muy descriptivo. Básicamente lo que estamos haciendo es crear una página con una etiqueta, cada vez que se abre la página revisamos si el usuario ya ha entrado al sitio, en caso de que sea su primera visita, aparece un mensaje como el de la *Figura P11.1* en la página:

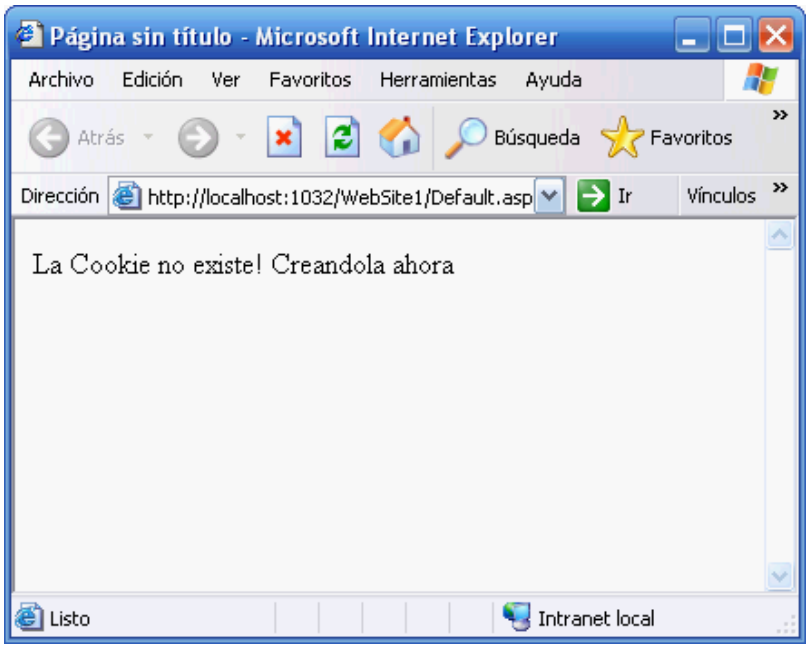


Figura P11.1

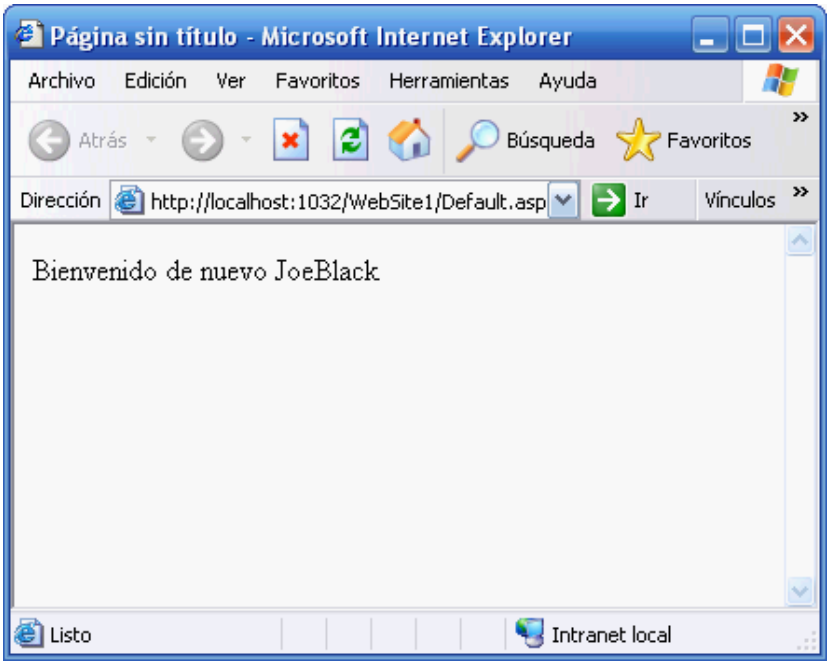


Figura P11.2

Al presionar el botón de actualizar (volver a visitar la página) aparece un mensaje como el de la - *Figura P11.2.* Nota que el tiempo de expiración de la cookie se estableció un mes después de esta fecha, por lo tanto, si se cierra la ventana del browser y se vuelve a abrir (o se vuelve a ejecutar la aplicación) nos volverá a presentar la imagen de la *Figura 11.2* debido a que la cookie sigue existiendo.

Actividades

- Implementar la funcionalidad de tu sistema que se encargará de crear una sesión cuando el usuario acceda al sistema.
- Implementar la funcionalidad de tu sistema que almacenará los datos de la sesión de un usuario en una *cookie*, para que los datos perduren aun cerrando la ventana del navegador.

Cuestionario

- ¿Qué ventajas tiene utilizar sesiones de usuario y cookies en una aplicación web?
- ¿Para qué utilizarías el objeto *Application* en tu sistema?

6.12 Práctica 12 - SQL Server y construcción de Bases de Datos

Objetivos

- Aprender a crear bases de datos en Visual Studio para usarlas con SQL Server.
- Conocer los elementos de programación que ofrece C# para el manejo de bases de datos.

Introducción

SQL Server

SQL Server es un manejador de base de datos relacionales (SGBD) creado por Microsoft que se ha integrado al entorno de desarrollo .NET junto con la tecnología ADO.NET, entre sus características figuran:

- Escalabilidad, estabilidad y seguridad.
- Soporta procedimientos almacenados.
- Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información.
- Además permite administrar información de otros servidores de datos

Este sistema incluye una versión reducida, llamada MSDE con el mismo motor de base de datos pero orientado a proyectos más pequeños, que en su versión 2005 pasa a ser el **SQL Express Edition**. Esta versión es con la que trabajaremos en este curso.

Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos como son Oracle, Sybase ASE, PostgreSQL o MySQL.

Desarrollo

La ventana **Explorador de Servidores** de Visual Studio te da acceso a la mayoría de las características relacionadas con las bases de datos. Para ver esta ventana selecciona ***ver -> Explorador de servidores*** en el menú principal. Da click derecho sobre el nodo **Conexiones de Datos** y selecciona **Agregar conexión** (*Figura P12.1*)**.**

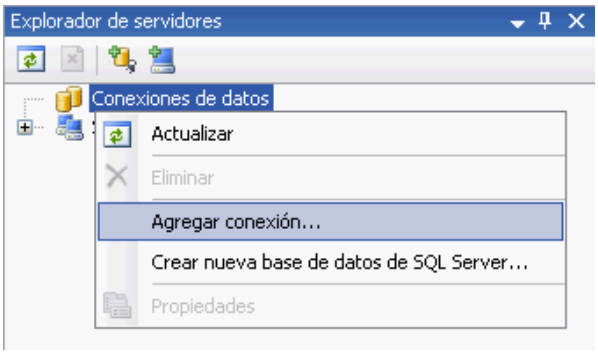


Figura P12.1

Selecciona Microsoft SQL Server en el cuadro de dialogo que aparece para elegir el origen de datos y presiona continuar. A continuación aparece la ventana Agregar conexión (*Figura P12.2*), realiza lo siguiente:

- Escribe **localhost\SqlExpress** como nombre del servidor.
- Deja la opción **Utilizar autentificación de windows** seleccionada.
- Escribe el nombre de la base de datos, en este caso el nombre es **Mibase**.
- Da click en **Aceptar**.

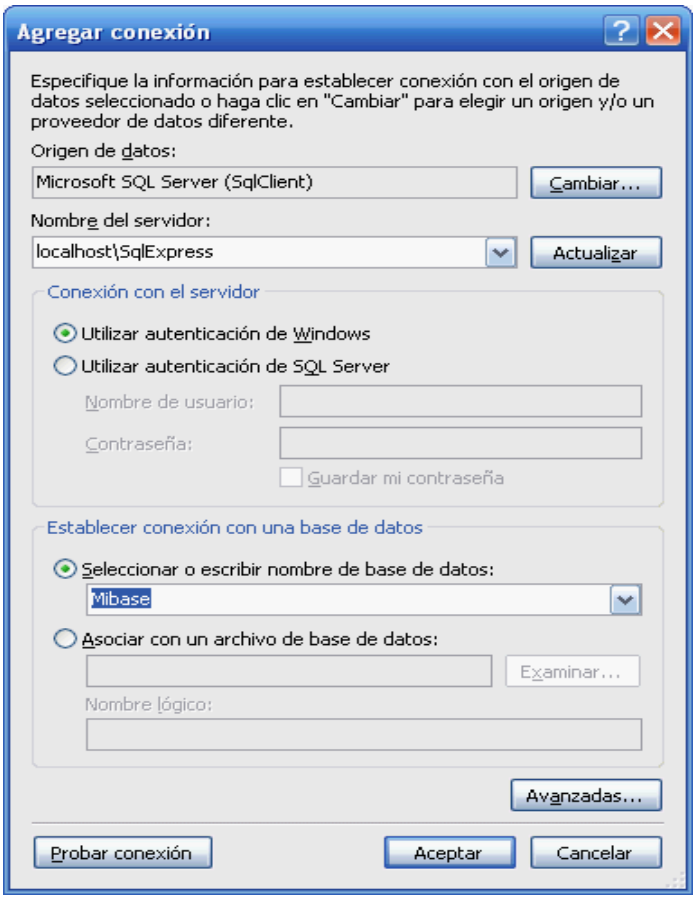


Figura P12.2

Se te pedirá confirmar la creación de una nueva base de datos llamada Mibase, da click en Sí. La nueva base de datos será creada, y se agregará un nodo bajo el nodo de **Conexiones de datos**, puedes expandirlo para ver su contenido (*Figura P12.3*).

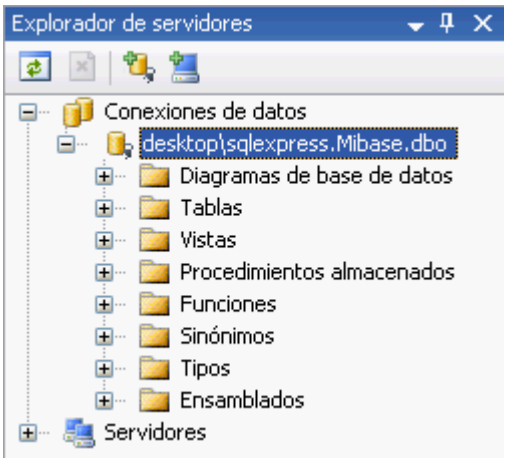


Figura P12.3

Crear Tablas

Para crear una tabla, selecciona el nodo **Tablas** que desciende del nodo de la base de datos que creaste, da click derecho y selecciona **Agregar nueva tabla** (*Figura P12.4*).

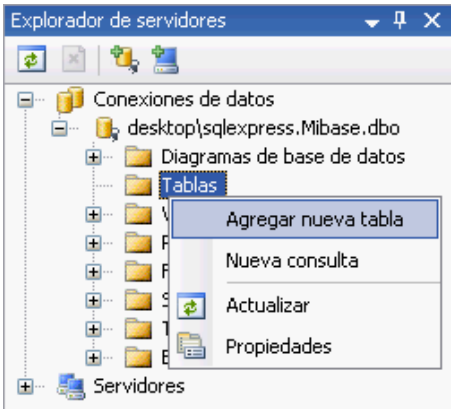


Figura P12.4

La ventana que aparece como resultado de este procedimiento se muestra en la *Figura P12.5*. Esta ventana de edición te permite crear columnas (campos) y especificar las tres propiedades principales de una columna: **Nombre de columna**, **Tipo de datos**, y **Permitir valores nulos**.

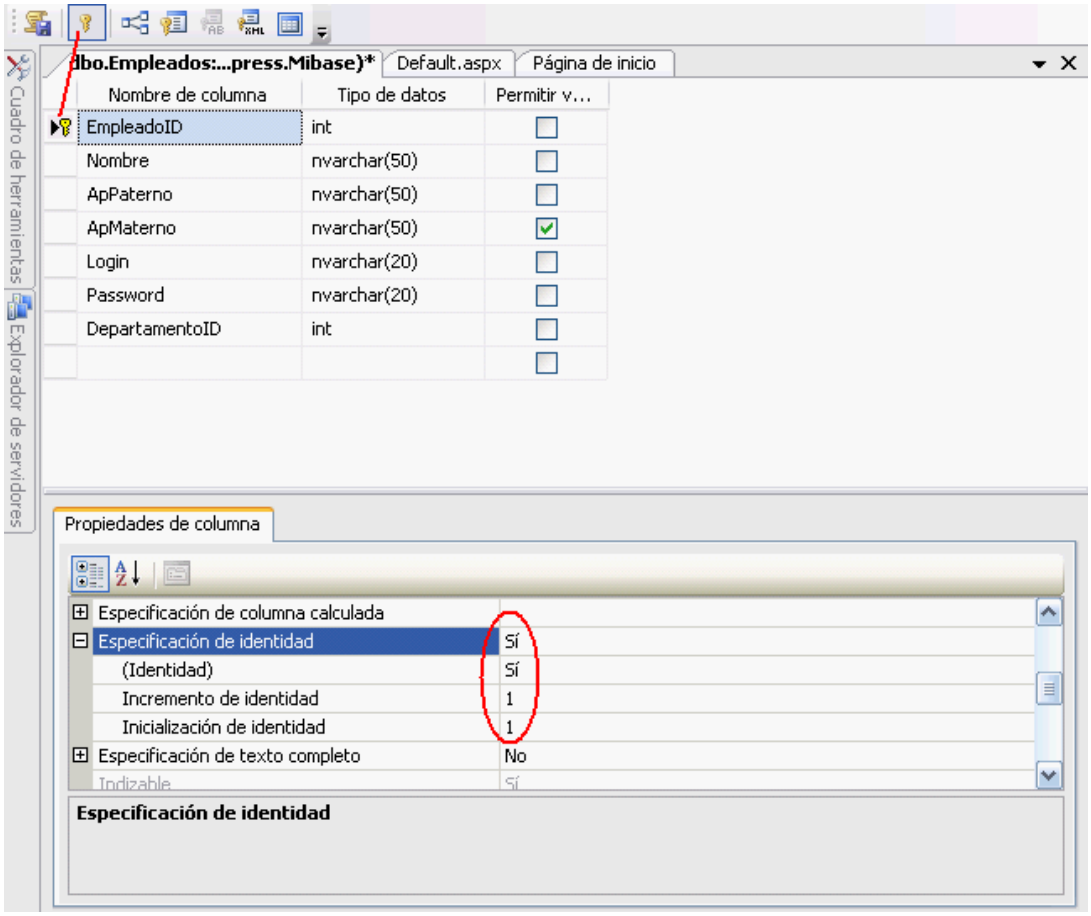


Figura P12.5

Para agregar la propiedad Identidad a una columna, selecciona la columna deseada, y en el panel **Propiedades de columna** localiza la propiedad **Especificación de identidad** y expándela. Esto revelará tres sub-secciones. Asigna **Sí** en la sección **(Identidad)** como se muestra en la *Figura P12.5*. Para establecer una columna como llave primaria, selecciona la columna y presiona el ícono en forma de llave en la barra de herramientas del diseñador de tablas, una pequeña llave aparecerá a un lado de la columna.

Después de introducir la información correspondiente de cada columna que conforma la tabla, presiona **Ctrl+S** para salvar la tabla, escribe el nombre de la tabla en el cuadro de dialogo que aparecerá (en este caso Empleados). Un nuevo nodo llamado **Empleados** aparecerá debajo del nodo **Tablas** en el **Explorador de servidores**. Si cierras la ventana del diseñador de tablas, puedes abrirla dando click derecho sobre el nodo de una tabla en el **Explorador de servidores** y seleccionado **Abrir definición de tabla**.

Agregar una llave foránea: Para ilustrar como manejar la relación entre tablas utilizaremos un ejemplo: Tenemos una tabla **Empleado** la cual contiene un campo llamado **DepartamentoID**, por otra parte tenemos una tabla llamada **Departamento** con un campo (que es llave primaria) llamado **DepartamentoID**. Para crear una llave foránea se realizan los siguientes pasos:

- Selecciona la columna de la tabla que tendrá la llave externa (foránea), presiona el botón **Relaciones** de la barra de herramientas del menú del diseñador de tablas, en nuestro ejemplo la tabla es **Empleados** y la columna es DepartamentoID (*Figura P12.6*).

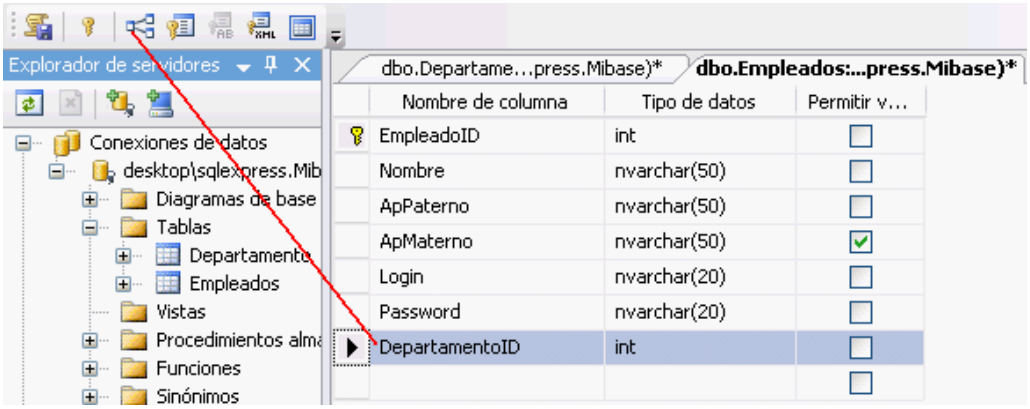


Figura P12.6

- Aparecerá el cuadro de dialogo **Relaciones de clave externa**, presiona el botón **Agregar** para agregar una relación, en el cuadro de propiedades selecciona la propiedad **Especificación de tablas y columnas**, y presiona el botón con los tres puntos para abrir un nuevo cuadro de diálogo (*Figura P12.7*).
- Selecciona la tabla que contiene la llave primaria original. En nuestro ejemplo es la tabla **Departamento** y selecciona los campos que conformarán la relación entre las tablas. En nuestro ejemplo los campos son **DepartamentoID** (*Figura P12.8*). Da click en Aceptar para volver a la ventana **Relaciones de clave externa**.

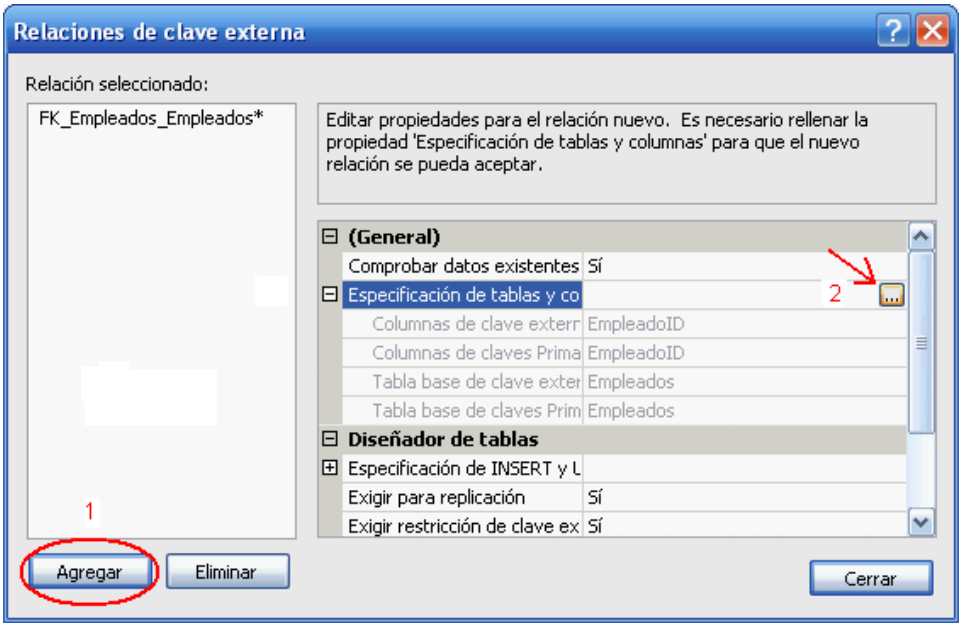


Figura P12.7

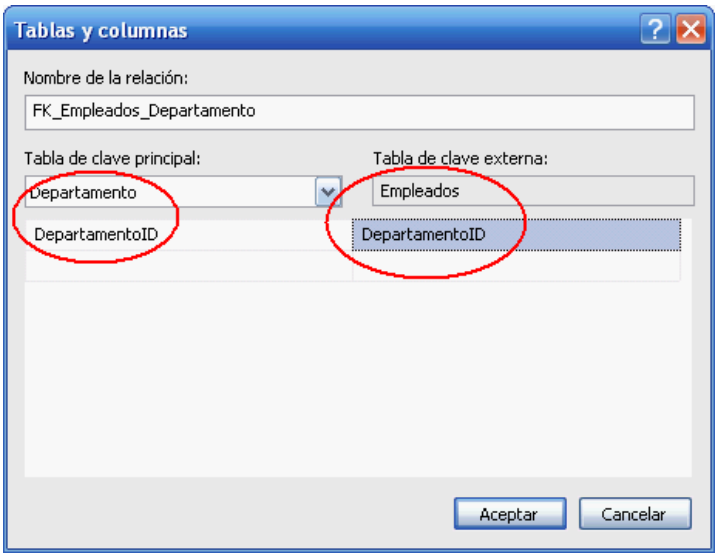


Figura P12.8

- En la ventana de **Relaciones de clave externa** puedes establecer las características que tendrá la llave foránea. Nota que la propiedad **Constraint** de las llaves foráneas está traducida como **Exigir restricción de clave externa**. Da click en **Cerrar,** la llave foránea se creará y las tablas quedarán relacionadas.

Diagramas de Base de Datos

Una gran característica de Visual Studio 2005, es que te permite ver el diagrama de la base de datos que estas creando. Para ver el diagrama de la base de datos que se ha creado en nuestro ejemplo, seguimos los siguientes pasos:

- Click derecho sobre el nodo **Diagramas de base de datos** y selecciona **Agregar nuevo diagrama** (*Figura P12.9*).

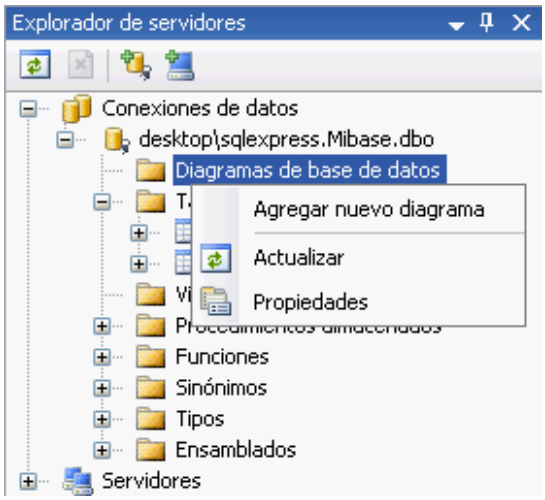


Figura P12.9

- Aparece un nuevo cuadro de dialogo en el que puedes elegir las tablas de la base de datos que quieres que aparezcan en tu diagrama. Esto es especialmente útil cuando trabajas con bases de datos cuyo diagrama es demasiado grande y solo te interesa ver una parte de él. Agrega ambas tablas seleccionando cada una y presionando Agregar (*Figura P12.10*).

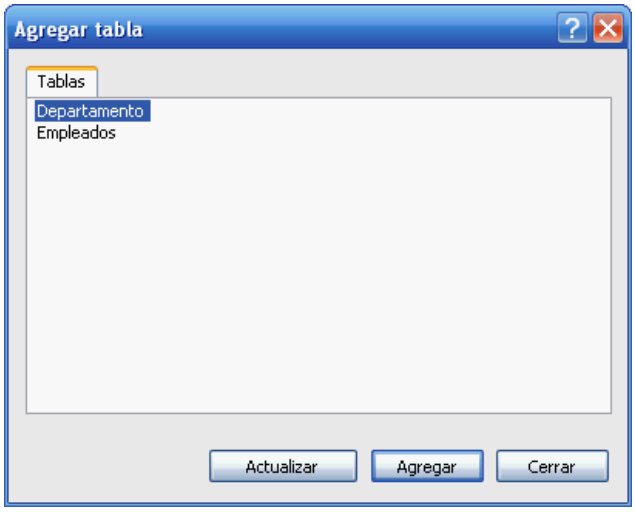


Figura P12.10

Desde el diagrama de bases de datos también puedes crear tablas y relacionarlas entre sí, de una forma mucho más intuitiva. En la *Figura 12.11* se muestra como quedó el diagrama de nuestro ejemplo.

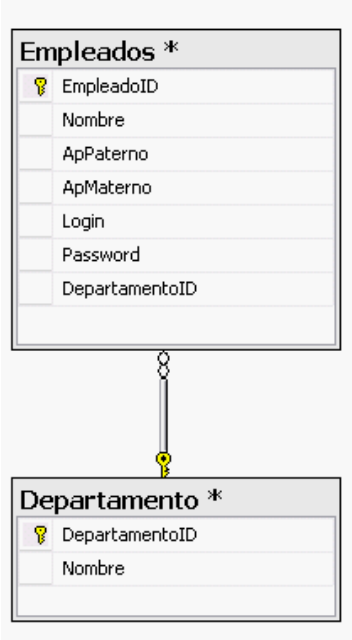


Figura P12.11

Actividades

- Realizar el diseño del diagrama de base de datos que utilizarás en tu aplicación Web, teniendo especial cuidado en las relaciones que existen entre cada tabla que conforma tu base de datos.

- Crear la base de datos en Visual Studio 2005.

Cuestionario

- ¿Qué ventajas y desventajas encuentras en crear una base de datos por medio de un administrador gráfico como el que provee Visual Studio?
- ¿Qué otros sistemas de gestión de bases de datos has utilizado además de SQL Server?, ¿Cuáles son sus diferencias?

6.13 Práctica 13 - Programación I/O para Bases de Datos con ADO.NET

Objetivos

- Conocer que es ADO.NET y cómo se utiliza.
- Aprender a programar la entrada y salida de información en bases de datos con C#.

Introducción

ADO.NET

ADO.NET es una tecnología que .NET ofrece para realizar el manejo de bases de datos a través del código de una aplicación. ADO.NET es capaz de usar diferentes tipos de conexiones a datos, dependiendo de a qué tipo de base de datos se esté tratando de conectar una aplicación. Las clases de ADO.NET cuyos nombres empiezan con ***Sql*** están específicamente construidas para conectarse con SQL Server. Existen clases similares para otros tipos de bases de datos, por ejemplo: si estás trabajando con **Oracle**, puedes utilizar clases como **OracleConnection**, **OracleCommand**, etc.

Para usar **ADO.NET**, se debe decidir qué tipo de base de datos se utilizará, así sabremos que **Namespaces** debemos importar, ya que estos deben contener las clases adecuadas para trabajar con la base de datos.

Dado que estamos utilizando **SQL Server**, importaremos el namespace **System.Data.SqlClient**, este contiene todas las clases ***Sql*** que requerimos. Las clases más importantes son:

SqlConnection: Esta clase contiene propiedades y métodos para conectarse a una base de datos de SQL Server.

SqlCommand: Esta clase contiene propiedades y métodos que permiten realizar consultas SQL, las cuales se llevarán a cabo sobre la base de datos en el servidor **SQL Server**.

SqlDataReader: Los datos de la base de datos son regresados en la clase SqlDataReader. Esta clase contiene propiedades y métodos que permiten iterar a través de los datos que contiene.

El namespace **System.Data.SqlClient** contiene muchas más clases que las tres mencionadas anteriormente, pero estas tres clases son capaces de resolver la mayoría de la tareas necesarias para leer y almacenar información en una base de datos.

Cuando se trabaja con **ADO.NET**, la tarea de establecer un enlace entre la base de datos y la aplicación es un proceso que sigue los siguientes pasos:

- Importar los Namespaces necesarios.
- Definir la conexión a la base de datos con un objeto de la clase **SqlConnection**.

- Cuando se requiera manipular la base de datos, establecer la consulta (**query**) apropiada en un objeto de la clase **SqlCommand**.
- Abrir la conexión y ejecutar la consulta SQL para guardar el resultado en un objeto de la clase **SqlDataReader**.
- Extraer los datos relevantes del objeto **SqlDataReader** y desplegarlos en la aplicación.
- Cerrar el objeto **SqlDataReader** al igual que la conexión a la base de datos.

Ejemplo

Para ilustrar el uso de las clases mencionadas anteriormente, utilizaremos un ejemplo sencillo: Nuestra pequeña aplicación obtendrá el nombre completo de un empleado por medio de un formulario y lo almacenará en una base de datos, además mostrará la lista de todos los empleados que han sido registrados.

La base de datos que utilizaremos está conformada por una tabla llamada **Prueba,** contiene una tabla **Empleados** y esta a su vez contiene los campos: **EmpleadoID**, **Nombre**, **ApPaterno** y **ApMaterno**, en la *Figura P13.1* se muestra como se ve la base de datos en el explorador de servidores.

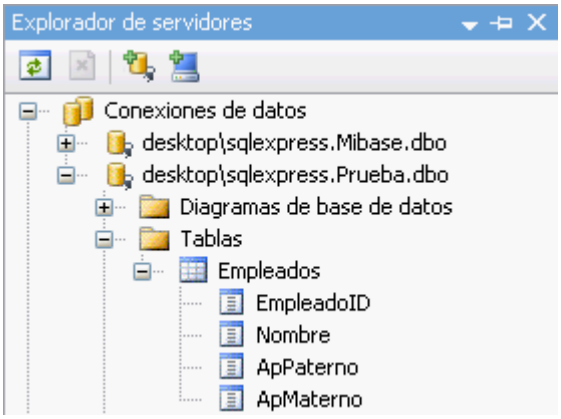


Figura P13.1

La interfaz de la aplicación consta de 3 cajas de texto, dos botones y una etiqueta (*Figura P13.2*).

Debido a las características de .NET, no es estrictamente necesario conocer como está conformado el código que crea la interfaz mostrada anteriormente, lo único que necesitamos saber es, cual es el nombre de los controles Web que se utilizan en la interfaz. Los nombres de las cajas de texto son: **Nombre**, **ApPaterno**, y **ApMaterno**. Los botones se llaman **Registra** y **Listar**. Por último la etiqueta se llama **ListaEmpleados**.

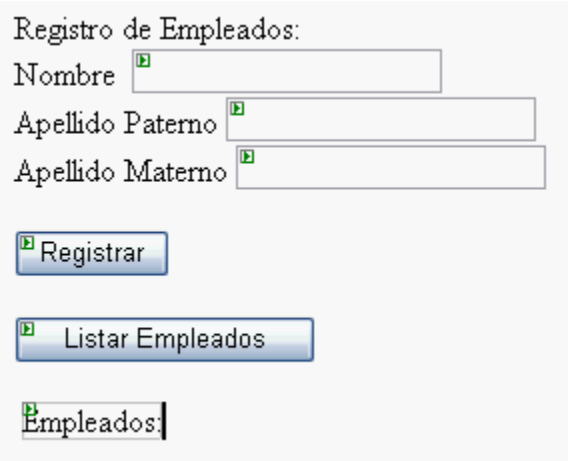


Figura P13.2

Demos un vistazo al código del archivo code-behind de la aplicación:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;

public partial class _Default : System.Web.UI.Page
{
    SqlConnection conexion;

    protected void Page_Load(object sender, EventArgs e)
    {
        conexion = new SqlConnection(
    "Server=localhost\\SqlExpress;Database=Prueba;" +
    "Integrated Security=True");
    }

    protected void Registra_Click(object sender, EventArgs e)
    {
        // Crea la consulta que queremos realizar
        SqlCommand consulta = new SqlCommand(
    "INSERT INTO Empleados VALUES ('" + Nombre.Text + "', '" +
    ApPaterno.Text + "', '" + ApMaterno.Text + "') ", conexion);

        // Abre la conexion
        conexion.Open();
        // Ejecuta la consulta
        consulta.ExecuteNonQuery();
        // Cerramos la conexion
        conexion.Close();
    }
```

```
    protected void Listar_Click(object sender, EventArgs e)
    {
        // Creamos la consulta que queremos realizar
        SqlCommand consulta = new SqlCommand(
    "SELECT Nombre, ApPaterno, ApMAterno FROM Empleados", conexion);
        conexion.Open();
        // Ejecuta la consulta y se guarda el resultado
        SqlDataReader resultado = consulta.ExecuteReader();
        // Inicializa el texto de la etiqueta
        ListaEmpleados.Text = "Empleados: <br>";
        // Lee el resultado y lo despliega
        while (resultado.Read())
        {
            ListaEmpleados.Text += resultado["Nombre"] + " " +
                resultado["ApPaterno"] + " " +
                resultado["ApMaterno"] + "<br>";
        }
        // Cierra el reader y la conexion
        resultado.Close();
        conexion.Close();
    }
}
```

A continuación se analizará el código parte por parte tomando en cuenta la serie de pasos que se enunciaron anteriormente:

Importar los Namespaces necesarios

Primero se debe importar el namespace que contiene las clases que necesitamos, esto se realiza en la línea:

```
using System.Data.SqlClient;
```

Definir la conexión a la base de datos

Para definir la conexión a la base de datos necesitamos crear un objeto de la clase SqlConnection, y pasar una **cadena de conexión** como parámetro en su constructor. Una **cadena de conexión** es una cadena en la cual se especifica la base de datos a la que queremos conectarnos, así como los datos de autentificación requeridos. Una cadena de conexión típica para SQL Server se ve así:

"Server=computer\SqlExpress; Database=database; User ID=username; Password=password"

La cadena de conexión debe especificar el nombre de la computadora en la cual se localiza la base de datos (puedes utilizar **localhost** para referirte a la maquina local), y el nombre asignado al servidor de base de datos (**SqlExpress** es el nombre para SQL Server Express), además se requieren el nombre de la base de datos (en nuestro caso **Prueba**), el password y ID del usuario.

SQL Server soporta dos métodos de autentificación: **SQL Server Authentification** y **Windows Authentification**. La forma de autentificación que utilizamos para crear la base de datos de esta

aplicación fue **Windows Authentification** (al igual que en la práctica pasada), para decirle a SQL Server que usamos autentificación con Windows, nuestra cadena de conexión debe presentar la cadena **Integrated Security=true**, en lugar de **username** y **password**, así:

Server=computer\SqlExpress; Database=database; Integrated Security=True

La parte del código que crea la conexión a la base de datos es esta:

```
SqlConnection conexion;

    protected void Page_Load(object sender, EventArgs e)
    {
        conexion = new SqlConnection(
    "Server=localhost\\SqlExpress;Database=Prueba;" +
    "Integrated Security=True");
    }
```

Observa que se crea un objeto llamado conexión, el cual es un atributo de clase, y que la conexión se define en el método Page_Load, se hace así para no tener que definir un objeto **SqlConnection** en cada método que se requiera hacer una conexión, de esta forma la conexión se define en el momento en que se abre la pagina.

Estableciendo la consulta

Para establecer la consulta se crea un objeto SqlCommand cuyo constructor acepta dos parámetros: El primero es una cadena que debe contener la consulta que se desea realizar, el segundo es el objeto de la conexión que creamos en el paso anterior.

En nuestra aplicación, realizamos una consulta a la base de datos cada vez que se presiona un botón, es por eso que se crea un objeto **SqlCommand** en el método del evento click de cada botón. En el método del botón **Listar** se presenta una consulta muy simple:

```
// Creamos la consulta que queremos realizar
        SqlCommand consulta = new SqlCommand(
    "SELECT Nombre, ApPaterno, ApMAterno FROM Empleados", conexion);
```

En cambio en el método del botón **Registra,** se presenta una consulta que está conformada por el texto contenido en cada caja de texto:

```
// Crea la consulta que queremos realizar
        SqlCommand consulta = new SqlCommand(
    "INSERT INTO Empleados VALUES ('" + Nombre.Text + "', '" +
    ApPaterno.Text + "', '" + ApMaterno.Text + "') ", conexion);
```

Abriendo la conexión y ejecutando la consulta

Cuando requieras obtener la información de la base de datos, debes abrir la conexión con el método Open de la clase SqlConnection y después debes ejecutar la consulta (comando). La clase SqlCommand cuenta con tres métodos para ejecutar un comando:

- **ExecuteReader:** Es utilizado para ejecutar consultas (queries) que regresan uno o más renglones de datos. **ExecuteReader** regresa un objeto **SqlDataReader** que puede usarse para leer los resultados del query uno por uno. **SqlDataReader** no puede utilizarse para actualizar datos ni para acceder a los datos en forma aleatoria. **SqlDataReader** mantiene la conexión a la base de datos abierta hasta que todos los registros sean leídos, por esa razón se debe cerrar la conexión con el método Close.

- **ExecuteScalar:** Se utiliza para ejecutar queries que regresan un solo valor. Este método regresa un objeto tipo **Object**, el cual debes convertir a un tipo específico de dato, dependiendo del tipo de dato que esperas recibir.

- **ExecuteNonQuery:** Este método se utiliza para ejecutar queries que insertan, modifican o actualizan datos. El valor que regresa será el número de renglones afectados.

En el método del botón **Listar,** usamos **ExecuteReader** porque la consulta regresa uno ó varios renglones:

```
conexion.Open();
// Ejecuta la consulta y se guarda el resultado
SqlDataReader resultado = consulta.ExecuteReader();
```

En cambio en el método del botón **Registra,** se usa **ExecuteNonQuery** porque la consulta inserta los datos que obtenemos del formulario:

```
// Abre la conexion
conexion.Open();
// Ejecuta la consulta
consulta.ExecuteNonQuery();
```

Obteniendo los datos y desplegándolos en la aplicación.

En el caso de que la consulta que realizamos, regrese algún dato que necesitamos desplegar en la aplicación, es necesario leer cada dato para poder mostrarlo en pantalla. En el método del botón **Listar** nuestra consulta regresa varios renglones, los cuales debemos leer uno por uno para desplegarlos en la página de la aplicación:

```
// Lee el resultado y lo despliega
while (resultado.Read())
{
```

```
    ListaEmpleados.Text += resultado["Nombre"] + " " +
    resultado["ApPaterno"] + " " +
    resultado["ApMaterno"] + "<br>";
}
```

Cerrando la conexión a la base de datos

Finalmente lo único que hace falta es cerrar la conexión a la base de datos y cerrar la conexión del lector de resultados (en caso de que exista). En el método del botón **Listar** (a diferencia del método del botón **Registra**) creamos un **Reader** llamado resultado, como mencionamos anteriormente esta conexión también debe cerrarse:

```
resultado.Close();
conexion.Close();
```

En la *Figura P13.3* se muestra la ejecución de nuestra aplicación. Como se puede observar cada botón realiza una acción que utiliza la base de datos para presentar o almacenar información.

En esta práctica se han presentado de forma general, los pasos que se siguen para crear un enlace entre una base de datos y una aplicación, sin embargo, ADO.NET ofrece muchas más características de las que se han presentado aquí. Para tener una mayor referencia de todas las capacidades de SQL Server y ADO.NET consulta los libros presentados en la bibliografía.

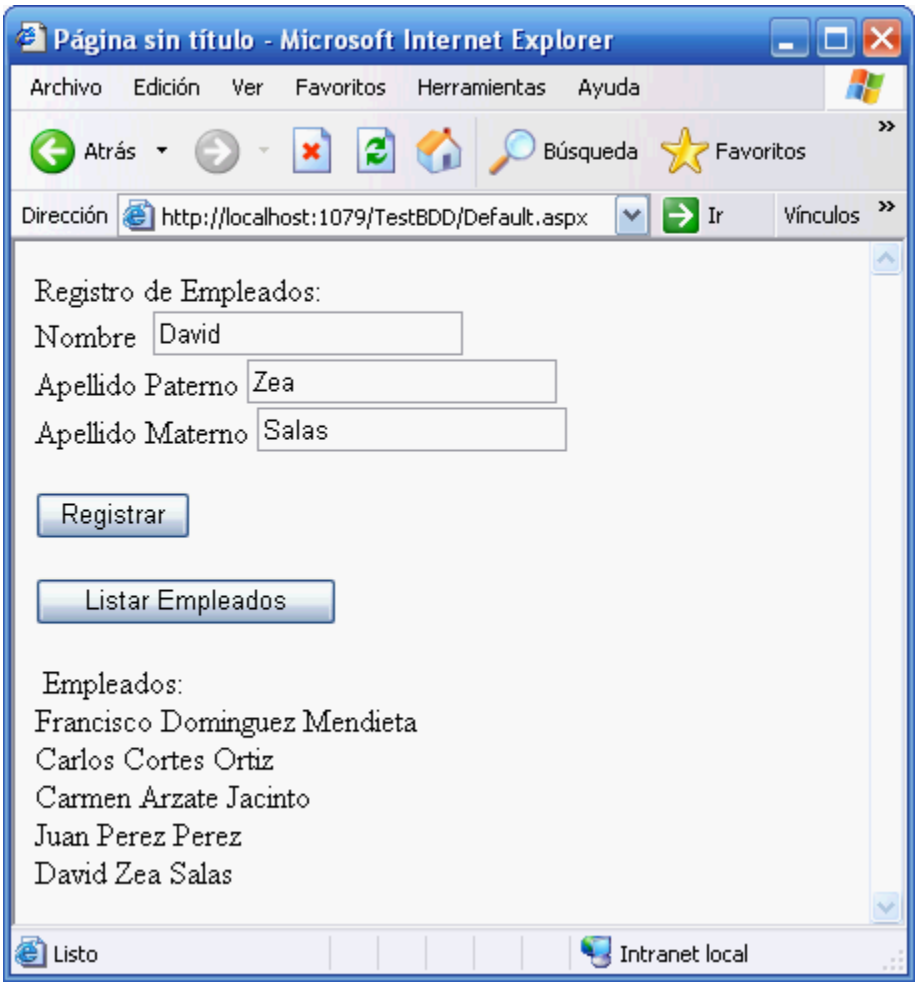


Figura P13.3

Actividades

- Implementar las partes de tu sistema que se encargan de crear la conexión con la base de datos.
- Implementar las partes de tu sistema que se encargan de guardar la información en la base de datos.
- Implementar las partes del sistema que se encargan de obtener la información de la base de datos.

Cuestionario

- ¿Con qué clase se realiza la conexión a una base de datos?
- ¿Para qué sirve la clase SQLCommand?

CAPÍTULO VII. Actualización de Herramientas

A la fecha en la que se realiza este reporte, Microsoft ha liberado una nueva versión de su framework: El Framework .Net 4 que viene junto con la nueva versión de su *IDE* Visual Studio 2010. Cabe destacar que Microsoft Framework .NET no había tenido novedades de desarrollo desde hacía aproximadamente 3 años. En este capítulo se hablará brevemente de los cambios más significativos de estas nuevas versiones.

7.1 Framework .Net 4

Hay una gran cantidad de mejoras y anexos en la nueva versión de .Net, desde la adaptación para sistemas operativos y procesos de 64 bits, hasta la creación de nuevas clases que simplifican la programación de procesos en paralelo, sin embargo, en lo que respecta a este proyecto, el cambio más significativo es la mejora que se realizó en la implementación del patrón de diseño *Modelo Vista Controlador* (*MVC*) para el desarrollo de aplicaciones Web.

El patrón de diseño *MVC* ha sido utilizado desde hace ya algunos años, sin embargo la forma en la que es adaptado a un entorno de trabajo es donde realmente radica su importancia. Esto fue demostrado por el Framework Ruby on Rails hace aproximadamente 4 años, cuando bajo sus premisas “No te repitas” y “Convención sobre configuración” probó que se podía desarrollar una aplicación web hasta 5 veces más rápido que con un framework convencional.

Tomando estos antecedentes se han agregado mejoras al manejo del patrón *MVC* en *ASP.NET*, permitiendo así que los desarrolladores web puedan compilar atractivos sitios web basados en estándares que son fáciles de mantener porque reduce la dependencia entre los niveles de aplicación mediante el modelo *MVC* (*Model View Controller*). *MVC* proporciona un control completo sobre el marcado de las páginas. También mejora la capacidad para realizar pruebas ya que admite de forma inherente el desarrollo orientado a pruebas.

Los sitios web creados con *MVC* de *ASP.NET* tienen una arquitectura modular. Esto permite a los miembros de un equipo trabajar independientemente en los diferentes módulos y se puede usar para mejorar la colaboración. Por ejemplo, los desarrolladores de software pueden trabajar en los niveles de modelo y controlador (datos y lógica), mientras el diseñador trabaja en la vista (presentación).

No es objetivo de este proyecto enseñar a los alumnos a desarrollar bajo el patrón *MVC*, ya que los fines de estas prácticas son educativos y para entender el funcionamiento del framework .NET no hay mejor forma que desarrollar las aplicaciones desde los cimientos. Sin embargo es altamente recomendable que después de adquirir un nivel medio de experiencia con el framework .NET, se haga uso del patrón MVC para desarrollar las aplicaciones Web.

7.2 Visual Studio 2010

Visual Studio 2010 nace con una larga lista de mejoras y nuevas características, las cuales tienen dos finalidades principalmente:

1. Dar soporte total a la versión 4.0 de la plataforma .NET.
2. Hacer más fácil y productiva la experiencia de desarrollo.

A continuación se presentan algunas de sus nuevas características:

- En algunos escenarios donde las aplicaciones son bastante complejas y los desarrolladores gustan trabajar con más de un monitor, Visual Studio 2010 proporciona la facilidad de distribuir las ventanas del entorno en los monitores requeridos.

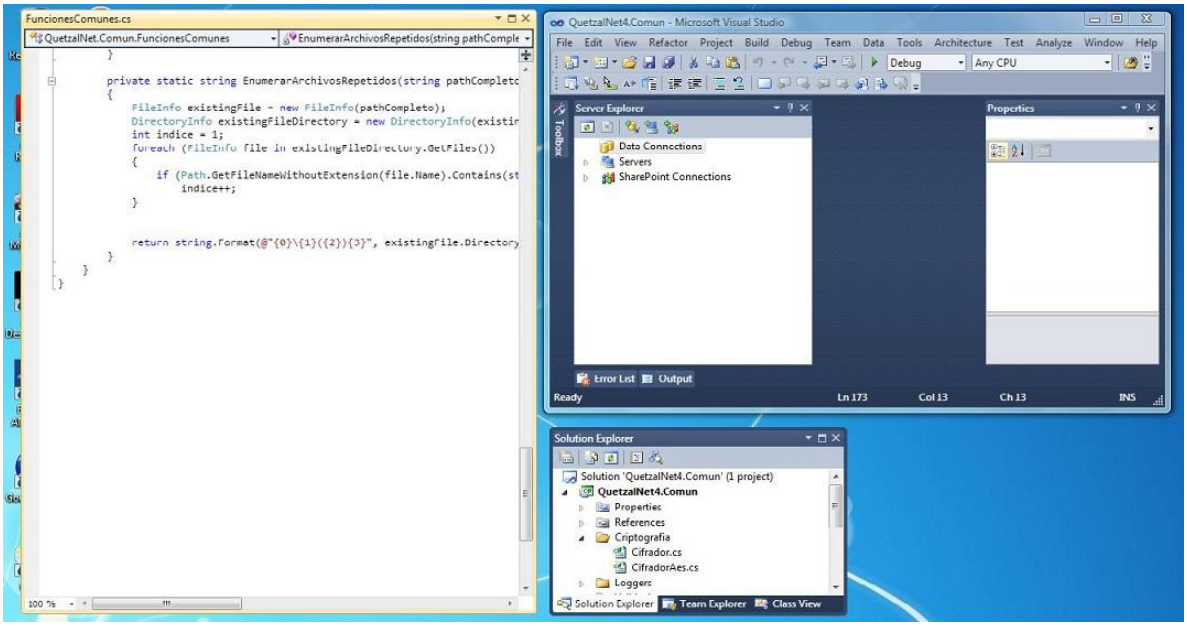


Figura 7.1

En la imagen anterior se aprecia como la ventana de código y la ventana del explorador de soluciones se encuentran fuera de la ventana principal de Visual Studio. Con una simple operación de arrastrar y soltar podemos colocar nuestras ventanas dentro y fuera de Visual Studio

- Una nueva característica modesta pero bastante útil, es permitir al desarrollador realzar ciertas partes de nuestro código simplemente colocando el cursor sobre alguna palabra referente a un tipo de dato, variable o método (*Figura 7.2*).

Figura 7.2

- Mejora de la experiencia del desarrollo de aplicaciones para Microsoft Office en sus versiones 2007 y 2010.
- Intellisense para JavaScript.
- Soporte total para proyectos Silverlight.
- Soporte a la administración de proyectos utilizando metodologías agiles.
- Nuevas características en el desplegado de aplicaciones Web (Web Packages, One-Click Publishing, Web Configuration Transformations).
- Mejoras a la experiencia de desarrollo utilizando Microsoft ADO.NET.
- Soporte para la programación en paralelo.
- Mejoras al desarrollo de aplicaciones SharePoint.
- Herramientas para el desarrollo de aplicaciones en la nube (Windows Azure).
- Soporte al nuevo lenguaje Visual F#.

Y la lista de mejoras podría continuar si ahondamos más en el *IDE*, sin embargo, las características mencionadas son algunas de las principales con las que como integrantes de un equipo de construcción de software podríamos utilizar en nuestro día a día.

Por último, es importante mencionar que como se muestra en las *Figuras 7.3* y *7.4*, el diseño de la interfaz de Visual Studio 2005 no ha cambiado mucho con respecto a su versión más reciente. Los menús relevantes, las barras de herramientas y las ventanas principales como lo son el ***explorador de soluciones***, ***la ventana de propiedades*** y ***el cuadro de herramientas***, mantiene su misma ubicación y forma, por lo tanto las prácticas presentadas en este proyecto (las cuales fueron realizadas en Visual Studio 2005) pueden seguirse incluso si se está trabajando con Visual Studio 2010.

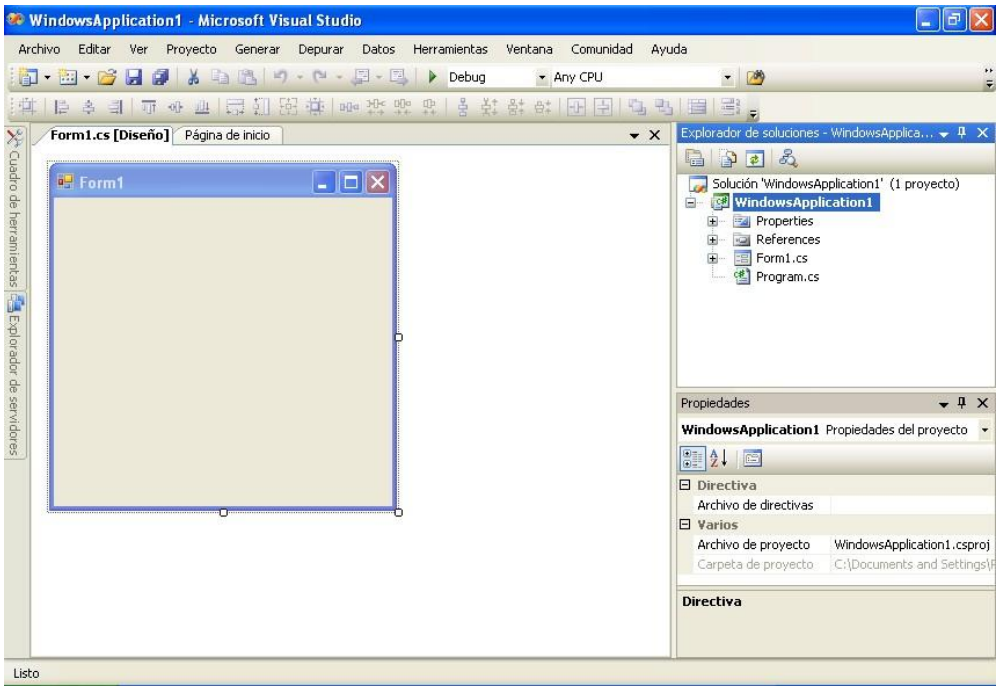


Figura 7.3. Visual Studio 2005

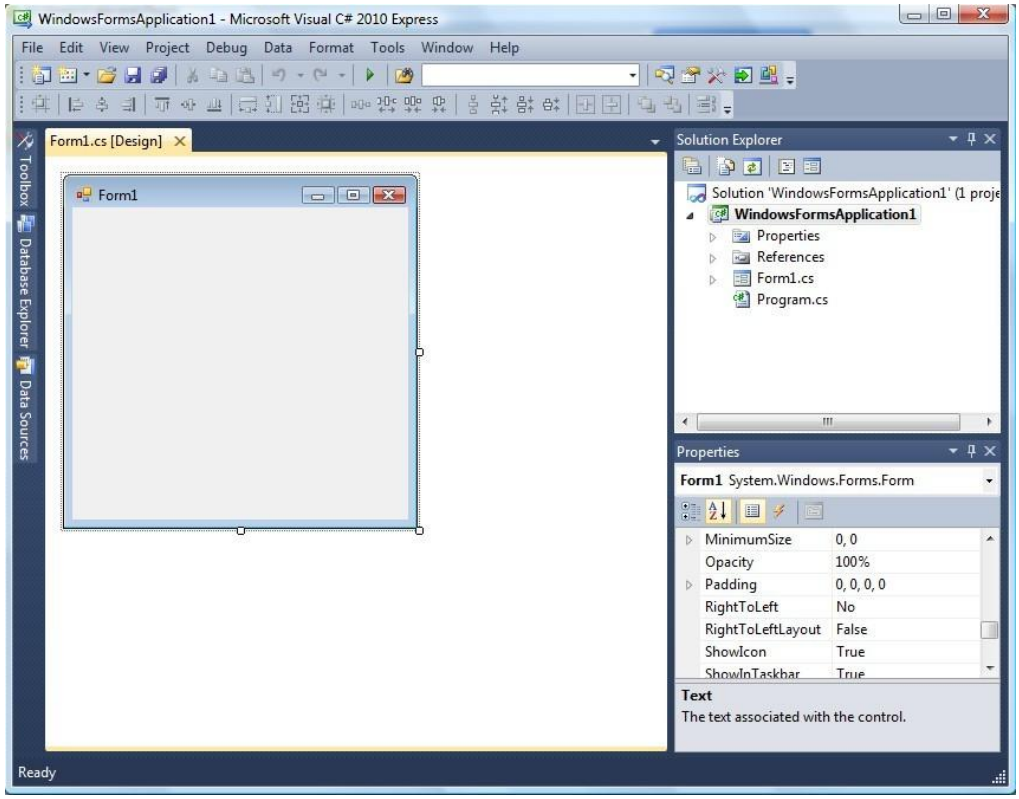


Figura 7.4. Visual Studio 2010

CONCLUSIONES

Los desarrolladores de software debemos estar preparados en la mayor cantidad de plataformas y lenguajes de programación posibles. Ya que esto nos proporciona una gran versatilidad para desenvolvernos en el mundo laboral y nos facilita el aprendizaje de nuevas tecnologías.

En un mundo perfecto, la elección de la plataforma de desarrollo dependería únicamente del desarrollador de software. Desafortunadamente para tomar esta decisión se deben tomar en cuenta otros factores que generalmente son determinantes, ya que en ocasiones no se desarrolla un sistema desde cero, sino que se deben crear nuevos módulos sobre un sistema ya existente, obligándonos a utilizar el lenguaje de programación en el cual fue desarrollado el sistema. El hardware también puede ser un factor determinante, ya que muchas plataformas de desarrollo son exclusivas del sistema operativo sobre el que funcionan y este a su vez está restringido al hardware.

En los últimos años el Framework .Net ha demostrado ser un entorno de desarrollo muy eficiente para crear aplicaciones que funcionan sobre el sistema operativo Windows, de igual forma con ASP.NET ha logrado competir de forma aceptable con el resto de las plataformas de desarrollo Web, como son Java, PHP y Ruby on Rails. Aunque más de uno pueda suponer que el éxito del Framework .Net se debe al monopolio de los productos Microsoft en el mercado, la realidad es que .Net es una plataforma confiable principalmente porque para su desarrollo se basaron en otras plataformas exitosas, como por ejemplo Java. Por esta razón los lenguajes de programación de .Net cuentan con todas las características necesarias para implementar el paradigma de programación orientada a objetos de manera eficiente. Incluso hay algunos estudiosos del tema que aseguran que los nuevos elementos agregados a este paradigma están mejor implementados en .Net que en Java, refiriéndose por ejemplo a la implementación de los tipos genéricos en ambas plataformas.

El uso del software libre nos trae indudables ventajas, como lo son la libertad de uso, modificación y distribución, además de la reducción de costos, sin embargo para las empresas, el software de propietario representa trabajar con la seguridad de no estar violando ninguna licencia o patente, además de sentir un respaldo, debido al servicio de soporte y garantía sobre fallas que el software libre no tiene. Por esta razón las empresas que se inclinan por utilizar software libre, y las que se inclinan por usar software de propietario, ocupan prácticamente el mismo porcentaje en el mundo empresarial.

Las practicas presentadas en este trabajo le enseñaran al alumno a crear software basándose en el proceso unificado, esta metodología ha sido aprobada por muchos desarrolladores de software a lo largo de los últimos años, en esta se señala que el desarrollo de un sistema debe ser guiado por los casos de uso, iterativo e incremental y centrado en la arquitectura. De esta forma se enseña a los alumnos que la documentación de un sistema juega un papel muy importante en el proceso de desarrollo de software. Los alumnos que decidan utilizar

estas prácticas para su desarrollo durante el curso de ciclo de desarrollo de software, tendrán una significativa ventaja sobre los demás, ya que no solo dominarán el lenguaje de programación Java el cual aprenden a lo largo de la carrera, sino que además tendrán el conocimiento necesario para desarrollar software en el Framework .Net, lo que duplicará sus oportunidades de conseguir una buena posición en el mundo laboral.

BIBLIOGRAFÍA

Richter Jeffrey. **Applied Microsoft .NET Framework Programming**. Microsoft Press A Division of Microsoft Corporation, 2005.

Lee Wei-Meng. **Practical .NET 2.0 Networking Projects.** Apress, 2006.

Troelsen Andrew. **Pro C# with .NET 3.0**, Special Ed. Apress, 2007.

Booch G., Rumbaugh J., Jacobson I. **The Unified Modeling Languaje**. Users guide. Addison Wesley, 1999

Darie Cristian, Ruvalcaba Zak. **Build Your Own ASP.NET 2.0 Web Site Using C# & VB**. SitePoint, 2006.

Foxall James. **Teach Yourself Microsoft® Visual C#® 2005 in 24 Hours**. Sams, 2006.

Filev Andrew, Loton Tony, McNeish Kevin, Schoellmann Ben, Slater John, G. Wu Chaur. **Professional UML with Visual Studio .NET - Unmasking Visio for Enterprise Architects.** Wrox Press, 2002.

Fowler Martin, Kendall Scott. **UML Gota a Gota.** Pearson, Addison Wesley, 1999.

Sommerville I. **Ingeniería de Software.** 6e ed. Addison wesley, 2002.

Pressman Roger, **Ingeniería del Software, Un enfoque práctico.** 6a Ed. McGraw Hill, 2005.

.Net Framework, Información general y conceptual. Microsoft Press 2005, http://msdn.microsoft.com/library/zw4w595w.aspx

Ibargüengoitia Guadalupe, López Gaona Amparo, Oktaba Hanna. **Notas del Curso de Ingeniería de Software.** Microsoft Faculty Resource Center 2008, https://www.facultyresourcecenter.com/curriculum/pfv.aspx?ID=7269&login=Ingenieria%20de%20Software