

Capítulo I Introducción a las Estructuras de Datos.

Contenido:

- Objetivo.
- Introducción.
- Estructuras de Datos y la Programación OO.
 - Que son las ED.
 - Qué es la POO.
 - Relación entre ED y POO.
- Conceptos de la Programación OO utilizando C Sharp y Visual Studio .NET como Entorno de Desarrollo.
- Conclusiones.
- Tarea.
- Clase Práctica

Objetivo.

- Aplicar los Conceptos Básicos de la POO utilizando C Sharp y Visual Studio como Entorno de Desarrollo.

Introducción.

Cuando trabajamos en una aplicación cualquiera, por ejemplo, un procesador de texto como Word, y se va la luz, se pierde el documento que hasta ese momento estemos escribiendo, si es que no lo hemos guardado en el disco duro. Esto ocurre debido a que estábamos escribiendo en la memoria interna de la Computadora. Por lo que para lograr esto, la memoria debió de ser organizada de forma tal que toda esa información se acceda de forma rápida y oportuna. Objetivo que se logra con el uso de las Estructuras de Datos. Aspecto a estudiar durante el transcurso de la asignatura.

Pero las aplicaciones que se desarrollan en el mundo real por lo general trabajan con grandes volúmenes de información que deben ser almacenados de forma permanente y se necesita contar con un soporte que permita preservar esta información y la memoria periférica es una buena alternativa.

Este capítulo se ocupa de introducir al estudiante en la Asignatura Estructura de Datos, conocer Qué es? y los conceptos de la programación orientada a objetos, ya que se programarán dichas

estructuras de datos, utilizando C Sharp y el enfoque orientado a objetos.

Que es una ED: Par (K,R).

- Se establece como un par K,R
- Donde K representa a diversos elementos y R las relaciones que se establecen entre ellos.
- Esto conlleva a diversos tipos de ED :
 - Lineales (cada nodo tiene un sucesor), ejemplo de Lista de Estudiantes.
 - Ramificadas, por ejemplo, un Director puede tener asociado a varios subdirectores, y a su vez estos tiene asociados varios Jefes de departamento).
 - Sobre cada una de estas ED, se pueden realizar diversas operaciones, entre ellas cabe citar a las operaciones de insertar y eliminar, vacía y llena.

Que es la POO.

El Mundo que nos rodea, es una colección de objetos que se relacionan entre sí.

El lenguaje natural está dotado de palabras, sustantivos, para nombrarlos y verbos para describir las operaciones o acciones que ellos desarrollan.

Es una técnica de estructuración, donde los objetos son los principales Elementos de Construcción que se conectan entre sí.

Es una Metodología de Programación de Propósito General que simula la forma en que el hombre trabaja.

Es una evolución natural de la programación. Un programa OO debe verse como un conjunto de Objetos en Comunicación.

Relación entre la POO y ED.

La clave de la programación OO es encontrar características comunes entre entidades implicadas en el problema y crear estructuras de datos (objetos) que capturen esas características comunes. Una vez identificados los objetos del problema, se definen sus características y se puede modelar su comportamiento.

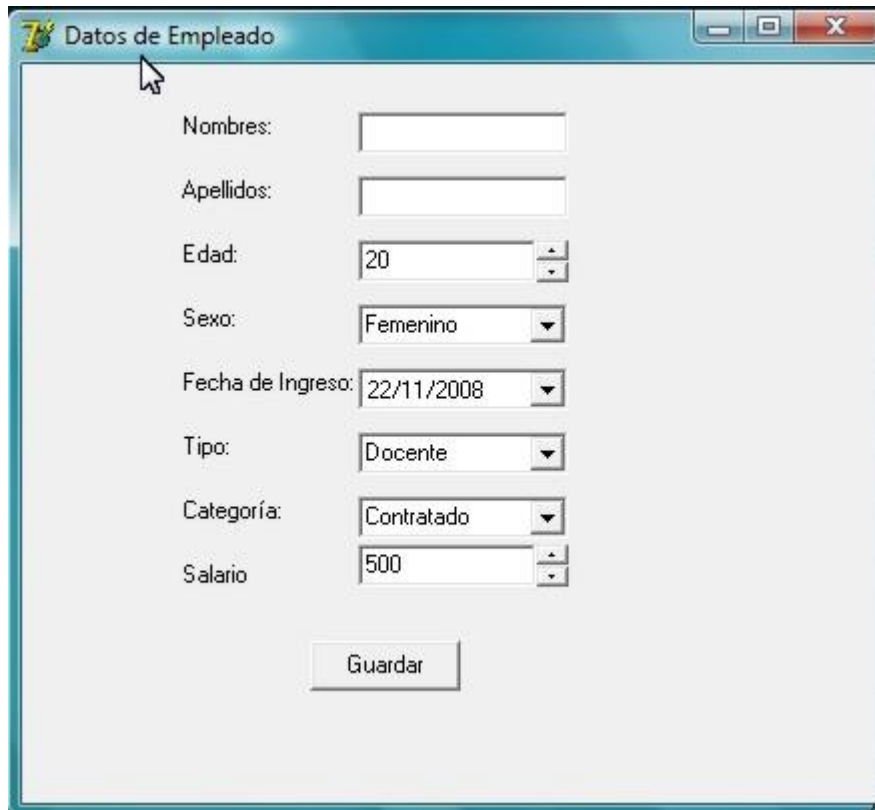
Conceptos Básicos de la Programación Orientada a Objetos

- Abstracción.
- Clase.
- Objeto
- Atributos:
- Métodos.

- Encapsulamiento.
- Instanciación.
- Polimorfismo

Para la descripción de los elementos de la POO, utilizaremos el siguiente ejemplo:

- Se quiere crear un programa para la manipulación de los recursos humanos de una Universidad, y se conoce la información de sus empleados Nombres, Apellidos, edad, Fecha de Ingreso , Categoría (contrato, fijo) y salario.
- Para comenzar veremos entonces la interfaz tentativa para resolver y representar la situación anterior:



The image shows a Windows application window titled "Datos de Empleado". The window contains a form with the following fields:

- Nombres:
- Apellidos:
- Edad: (spin box)
- Sexo: (dropdown menu)
- Fecha de Ingreso: (calendar picker)
- Tipo: (dropdown menu)
- Categoría: (dropdown menu)
- Salario: (spin box)

A "Guardar" button is located at the bottom center of the form.

- A continuación estudiamos y ejemplificamos los conceptos de la programación OO.

Abstracción.

Es aquella característica que nos permite identificar un objeto a través de sus aspectos conceptuales. Las características de los objetos pueden ser tan diferentes que puede costarnos reconocer que pertenecen a una misma clase, sin embargo nosotros reconocemos a que clase pertenecen, gracias a la Abstracción.

Ejemplos:

Un auto deportivo y otro familiar, son autos, ambos tiene rueda, volante, motor, puertas, etc.

Para el caso de estudio anterior, se tiene empleados administrativos y docentes, pero ambos gracias a la abstracción podemos decir que son Empleados

Un objeto **Empleado** para gestionar el personal de la empresa, aunque se tengan diferentes tipos de empleados.

Clase.

Es el conjunto de especificaciones o normas que definen como va a ser creado un objeto de un tipo determinado, algo parecido a un manual de instrucciones conteniendo indicaciones para crear el objeto.

Una clase constituye la representación abstracta de algo, mientras que un objeto constituye la representación concreta de lo que una clase define.

Determina el conjunto de puntos claves que ha de cumplir un objeto para ser considerado perteneciente a dicha clase o categoría, pues dos objetos de una misma clase no tiene que ser iguales, basta que cumplan las especificaciones de la clase.

Ejemplo:

Public Class TEmpleado

```
{  
    .....  
    .....  
    // Código de la clase.....  
  
    .....  
    .....  
}
```

Objeto.

Es la Unidad Básica de la Programación Orientada a Objetos.

Es una pieza que se ocupa de desempeñar un trabajo concreto dentro de una estructura organizativa de varios objetos, cada uno de los cuales ejerce la tarea particular para la que fue diseñada.

Es una agrupación de código, compuesta de propiedades y métodos que pueden ser manipulados como una unidad independiente.

Un objeto es un ejemplar de una clase.

Definición de un objeto en C Sharp:

```
TEmpleado Emp1; // Se define un objeto llamado Emp1, de tipo TEmpleado
```

Atributos Pasivos o Campos.

Definen los Datos del objeto permitiendo consultar o modificar su estado, en cualquier momento de acuerdo al procesamiento que se esté realizando.

Pueden ser de cualquier tipo, incluyendo al tipo Class, o sea, que un campo de un objeto puede ser un objeto de otra clase.

La declaración se realiza especificando un identificador y anteponiendo el tipo de datos.

Public Class TEmpleado

{

.....

string FCI, FNombres, FApellidos ;

string FDirección;

float FSalario;

Date Fecha ;

.....

}

Atributos Activos ó Métodos.

- 1.- Son las Rutinas que definen el comportamiento de la clase
- 2.-Se declaran a continuación de los campos.

Constructores

- ✓ Solicitan y reservan espacio en memoria
- ✓ Se utiliza el mismo nombre que la clase.
- ✓ Colocan un puntero a la zona de memoria donde están los métodos de la clase a la que corresponde el objeto, de manera que los mensajes que este reciba puedan ser atendidos adecuadamente.

Por ejemplo:

Public Class TEmpleado

```
{
string FCI, FNombres, FApellidos ;
string FDirección;
float FSalario;
Date Fecha;
....
Public TEmpleado (Strings CI, Strinbg aNombre, String aApellidos, integer aEdad, :string aSexo ,
Date aFechaIng, string aCategoria, integer aSalario, string aTipo);
{
    FCI= sCI;
    FNombre=aNombre;
    FApellidos=aApellidos;
    FEdad= aEdad;
    FSexo= aSexo;
    FFechaIng= aFechaIng;
    FCategoria=aCategoria;
    Salario=aSalario;
    Tipo=aTipo;
}
....
}
```

Procedimientos y Funciones

Se declaran utilizando el tipo que devuelve y el nombre del procedimiento o función, además la lista de parámetros que debe tener.

Por Ejemplo:

- Supongamos que se quiere manipular la información de tal manera que se pueda subir la categoría de un profesor a su inmediata superior, y además que en aquellos

casos en que no se exceda el salario máximo que es de 15000, esta operación implica el aumento de 300 pesos al mismo, cuyo monto mínimo es de 500.

```
Public Class TEmpleado
```

```
{
```

```
.....
```

```
string FCI, FNombres, FApellidos ;
```

```
string FDirección;
```

```
float FSalario;
```

```
Date Fecha;
```

```
String categoría;
```

```
Public TEmpleado()
```

```
{
```

```
....
```

```
}
```

```
Public string Nombre_Empleado()
```

```
{
```

```
Return nombre
```

```
}
```

```
Public SubirCategoria(string fijo);
```

```
{
```

```
    if FCategoria != Fijo
```

```
    {
```

```
        FCategoria = Succ(FCategoria);
```

```
        if FSalario + 1000 > 15000
```

```
            FSalario = 15000
```

```
        else
```

```
            FSalario = FSalario + 1000;
```

```
    }
```

```
}
```


Encapsulamiento.

Establece la separación entre el interfaz del objeto y su implementación.

Por Ejemplo:

Auto: Cuando se acelera, uno se limita a sólo eso, no se requiere saber la mecánica interna que hace que el auto se mueva más rápido o no.

En un sistema de gestión donde se necesite añadir un empleado, solamente se invoca al método añadir y él se encargará de hacer ese trabajo, no se necesita saber el código del método añadir.

Ventajas del encapsulamiento:

- Permite ignorar detalles de almacenamiento: A envía un mensaje a B y para ello no necesita conocer los datos de B, ni como se almacenan estos.
- Simplifica el programa. Cada objeto puede ser modificado sin afectar los módulos que lo acceden. Garantiza la integridad de los datos: el acceso a la información almacenada en un objeto, o sea, la recuperación, inicialización, actualización o modificación de dicha información, debe realizarse siempre a través de los métodos definidos en la clase a la que pertenece dicho objeto.
- Proporciona seguridad al código de la clase.
- Simplifica la utilización de los objetos. Ya que un programador que use un objeto, no necesitará conocer los detalles de su implementación, se limitará a utilizarlo.

Visibilidad de los Atributos.

- Cada atributo tiene una característica llamada visibilidad.
- La misma establece el acceso al mismo.
- La visibilidad indica cuáles atributos pueden ser manipulados desde el exterior del objeto, y quiénes pueden hacerlo.
- Se utilizan las palabras reservadas:
 - ✓ Private
 - ✓ Public.

- Cada una de las cuales representa un nivel de acceso al atributo.
- A continuación se explican las características de los elementos públicos, privados y protegidos.

Atributos Privados (Private).

- Un atributo privado no puede ser accedido desde fuera de la clase a la que pertenece.
- Es de uso privado de esa clase.
- Deben declararse como privados todos los atributos pasivos de una clase, como una forma de no violar el encapsulamiento.
- Ejemplo:

Atributos Públicos (Public)

- Un atributo público puede ser accedido desde cualquier clase o desde cualquier módulo donde la clase a la que pertenece pueda ser referenciada.
- Por omisión todos los atributos son públicos, si no se especifica visibilidad alguna, se asume que el atributo es público.
- Cuando un atributo no tiene especificado un nivel de visibilidad, se asume que tiene la misma visibilidad del atributo que le precede en la declaración de la clase.
- Dentro de cada nivel de visibilidad especificado en una clase, deben declararse siempre primero los atributos pasivos y luego los activos.
- Para mayor claridad en el código que se escribe, lo mejor es colocar todos los elementos privados juntos, seguidos de todos los protegidos y así, sucesivamente.

Instanciación.

- Es el hecho de crear un objeto como tal, con características propias.

Polimorfismo

- La palabra polimorfismo se deriva del griego poli: muchos y morfismo: forma, modalidad.
- Determina que el mismo nombre de método, realizará diferentes acciones según el objeto sobre el que sea aplicado.
- Ejemplo real:

Pelota y Vaso de Cristal: Si ejecutamos sobre ellos el método dejar caer, el resultado en ambos casos será diferente, mientras que el objeto pelota rebotará al llegar al suelo, el vaso cristal se romperá.

- Ejemplo de programación: Supongamos los objetos Ventana y Fichero, si ejecutamos sobre ambos el método abrir el resultado en ventana será la visualización de una ventana en el monitor, mientras que fichero se tomará un fichero en el equipo del usuario y se dejará listo para realizar sobre él operaciones de lectura o escritura.
- Ventajas del Polimorfismo: Permite generar componentes reutilizables de alto nivel que pueden adaptarse para que se ajusten a diferentes aplicaciones mediante el cambio de sus partes de bajo nivel.

Relaciones entre Objetos.

- Herencia.
- Pertenencia.
- Utilización.

Herencia:

- Mecanismo a través del cual se pueden construir clases en función de otras clases ya definidas.
- Establece que partiendo de una clase a la que denominamos clase base, padre o superclase, creamos una nueva clase denominada clase derivada, hija o subclase.
- En esta clase derivada disponemos de todo el código de la clase base, mas el nuevo código propio de la clase hija que escribamos para extender sus funcionalidades.
- A su vez podemos tomar una clase derivada y crear nuevas subclases a partir de ellas y así sucesivamente, componiendo lo que se denomina **jerarquía de clases**.
- En los lenguajes OO la herencia de atributos se extiende tanto a los datos o campos, como al comportamiento, o sea, los métodos.
- Se puede identificar a través del uso de la partícula **Es-un**.

Este Documento es de uso personal e intransferible, no se permite la reproducción total o parcial de este documento por ningún medio.

- Por ejemplo, sean los objetos A y B si tiene sentido el enunciado “A es un B” se puede diseñar la clase A como una subclase de B.
- Ejemplo:

Supongamos la siguiente situación:

Se tiene Clase Base Empleado, pero surge la necesidad de realizar pago a empleados que no trabajan en la empresa, pues son comerciales que se pasan la mayor parte del tiempo desplazándose, y para realizar esto, utilizamos Internet, necesitando el número de tarjeta de crédito, la dirección de email, etc.

Esto se puede resolver creando la clase derivada TCiberEmpleado que heredará de la clase empleado y en la que solo se debe añadir las nuevas propiedades y métodos para las transacciones comerciales, ya que las demás operaciones se tienen en la clase TEmpleado de la cual ella hereda.

Quedando La interfaz para trabajar con este problema de la siguiente manera:

Datos de Empleado

Nombres:

Apellidos:

Edad:

Sexo:

Fecha de Ingreso:

Tipo:

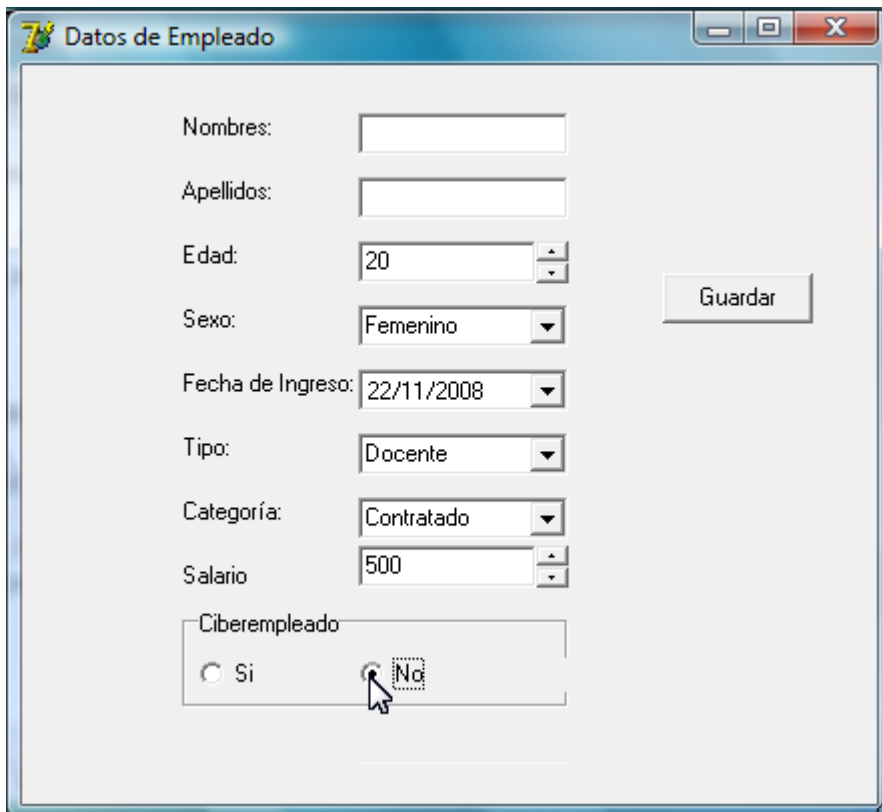
Categoría:

Salario:

Ciberempleado: ☒ Si ☐ No

Tarjeta:

Guardar



Pertenencia:

Los objetos pueden estar formados a su vez por otros objetos.

Se dice en este caso que hay una relación de pertenencia puesto que existe un conjunto de objetos que pertenecen a otro objeto o se unen para formar otro objeto.

A este tipo de relación se le llama también contenedora.

Para reconocer esta relación entre objetos se puede realizar un análisis sintáctico sobre la misma usando la partícula **(tiene - un)**.

Tiene-un también se puede expresar con Es-parte-de que se da también cuando alguno de los conceptos es un componente del otro, pero cuando los dos no son en sentido alguno, la misma cosa.

Tomando la interfaz vista anteriormente, podemos crear una nueva. Adicionando los controles necesarios para poder introducir la información referente a la dirección del empleado, ya sea este ciberEmpleado o no.

La interfaz para trabajar con este problema puede quedar de la siguiente manera:

Datos de Empleado

Datos Personales

Nombres:

Apellidos:

Edad:

Sexo:

Dirección

Calle:

Numero:

Ciudad:

Datos del Trabajo

Fecha de Ingreso:

Tipo:

Categoría:

Salario:

Ciberempleado

☐ Si ☒ No

Guardar

Utilización:

- Hay situaciones en que un objeto utiliza a otro para realizar una determinada tarea, sin que ello suponga la existencia de una relación de pertenencia entre dichos objetos.
- Para saber si existe esta relación debemos realizar un análisis sintáctico sobre el mismo empleando la partícula “usa un”
- El código de una clase hace uso (utiliza) un objeto de otra clase.
- Por ejemplo
- “Un objeto ventana usa un objeto empleado”, devolvería verdadero.
- Ejemplo:

Propiedades.

Una propiedad no es más que un atributo pasivo ficticio, que permite el acceso (lectura y escritura) a los campos de la clase, de manera que estos puedan ser siempre manipulados a través de ella.

- ✓ Los atributos pasivos de una clase pueden dividirse en campos o propiedades, que facilitan el trabajo de comunicación con los atributos pasivos.
- ✓ La función **GetNombreAtributo** para leer el valor del atributo
- ✓ El procedimiento **SetNombreAtributo** para escribir un valor en ese atributo.

Actividad Práctica

Habiendo leído el tema, y los diferentes conceptos de la Programación Orientada a Objetos. Ponga un ejemplo, donde utilice las propiedades utilizando C Sharp y Visual Studio .NET.

Nota: Hacer de 2 planas como máximo.

Se evaluará:

- Veracidad de los planteamientos.
- Claridad en el trabajo.