Práctica 2: AutoCAD como base de datos

Autor Miguel Ángel Alanis Montes

Objetivos

- Conocer la estructura de un software de CAD
- Aprender a utilizar transacciones en AutoCAD
- Dibujar entidades en AutoCAD con el API .NET

Software de CAD

Un software de CAD se enfoca en realizar representaciones gráficas de objetos físicos ya sea en segunda o tercera dimensión aplicados a un proceso de diseño.

Entre las características de un software de CAD tenemos

- Motor gráfico
- Definición Interactiva del objeto
- Modelado de tipos de objetos
- Información guardada con estructura de datos
- Generación de documentación a partir de la geometría
- Visualización

Base de datos de AutoCAD

AutoCAD guarda los objetos de forma jerárquica, siendo *DBObject* el objeto más sencillo en la Base de datos, cada objeto cuenta con un identificador llamado *ObjectId*, similar a una llave primaria en una Base de datos relacional.

Las características de un *ObjectId* son:

- Único por sesión o instancia de AutoCAD
- Se genera de manera automática al agregar un elemento a la base de datos.
- Se utiliza para acceder al objeto mediante una transacción
- Inconsistente puede reciclarse

Además de un *ObjectId* los elementos tienen otro identificador, llamado *Handle*.

Las características de un *Handle* son:

- Valor Hexadecimal con tamaño *long*
- Asociado a un *ObjectId*
- Único por archivo DWG.
- Autoincrementado al realizar una operación en la BD
- Consistente no se recicla.

Los objetos son expuestos en el espacio de nombre, *Autodesk.AutoCAD.DatabaseServices*

El objeto principal en el API de AutoCAD será *Application*, como su nombre lo dice hace referencia al programa de AutoCAD. La aplicación accede a los archivos abiertos mediante la propiedad *DocumentManager* la cual es de tipo *DocumentCollection*.

La clase *DocumentCollection* maneja la colección de archivos abiertos en la sesión actual de AutoCAD (*Application*) cuando queramos acceder al documento activo (*Document*) se usará la propiedad *MdiActiveDocument*.

Nos centraremos en la propiedad *Database* del documento, el cual contiene acceso a toda la información guardada en el archivo dwg.

Para acceder a la Base de datos del documento activo se usará la siguiente forma.

```
Database dwg = Application.DocumentManager.MdiActiveDocument.Database;
```

La estructura de AutoCAD puede ser revisada en la Ilustración 1.

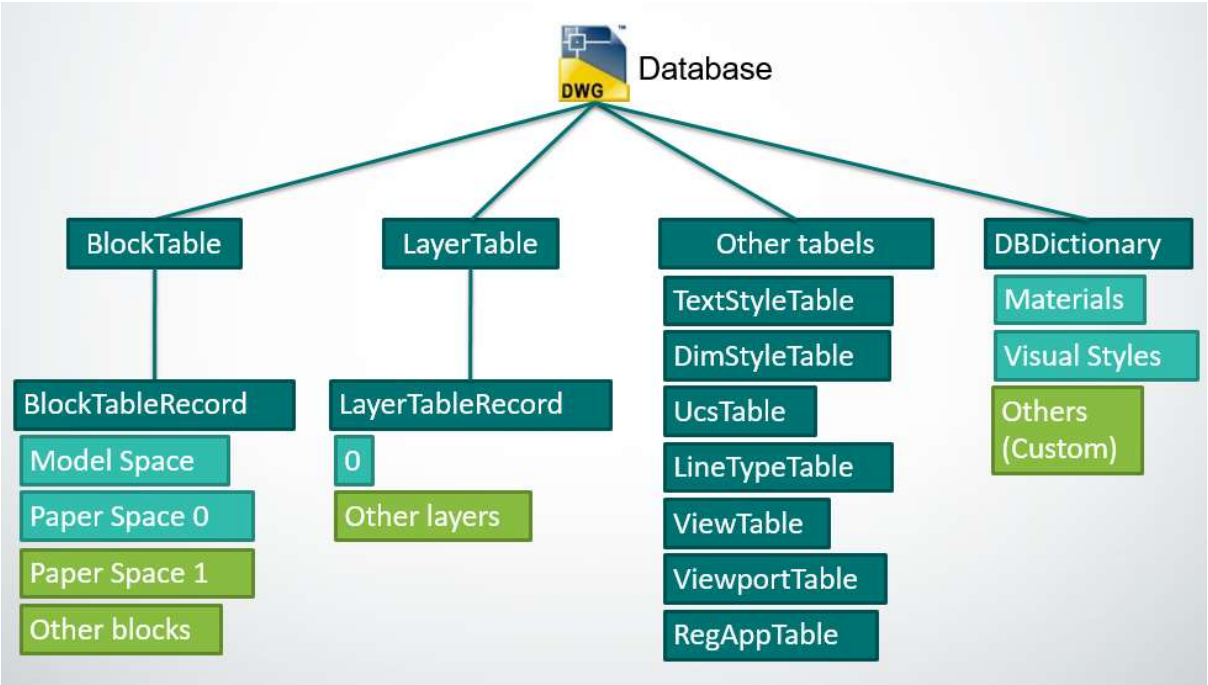


Ilustración 1 Diagrama de la base de datos de AutoCAD

En segunda nivel se tiene los objetos de tipo Tabla, por ejemplo:

- *BlockTable*
- *LayerTable*
- *TextStyleTable*
- *DimStyleTable*

Los objetos de tipo tabla heredan de la clase *SymbolTable* y su objetivo es organizar los registros (*SymbolTableRecord*). Los registros se asocian a una llave (*String*) y un id (*ObjectId*)

Las operaciones que se realizaran en la tabla son:

- **Add**: Agrega un nuevo registro en la tabla
- **Has**: Checa si un registro existe
- **this[ObjecId]**: Accede al registro por su Id
- **this[String]**: Accede al registro por su llave.

Las tablas son iterables y solo podrán guardar información de un tipo específico. La tabla de capas (*LayerTable*), solo puede guardar capas (*LayerTableRecord*).

En el tercer nivel de la Ilustración 1 se tienen los registros de la tabla, en el caso de la tabla de bloques al crear un nuevo archivo de AutoCAD tenemos siempre estos dos registros (*BlockTableRecord*).

- **Model Space**: En este registro se trabaja normalmente en AutoCAD
- **Paper Space 0**: En este registro se define un área de impresión.

Todo objeto en la base de datos se accede mediante una transacción.

Transacciones

1. Investigar que es una transacción en el ámbito de las tecnologías de distribución

Una transacción nos servirá para marcar los límites de una operación a la base de datos y a manejar excepciones de una manera más limpia.

2. Investigar los siguientes términos (tecnologías de distribución)

- Commit
- Abort
- Rollback
- Close (importancia)

Una transacción tiene las siguientes características

- Para guardar los cambios se debe mandar la operación *Commit*
- Para cancelar la operación podemos realizar un *Abort*
- Cerrar la transacción *Dispose*

La transacción será controlada usando la estructura *Using* y un bloque *Try Catch*.

3. Investigar cómo funciona el statement using y el bloque try catch

Las transacciones son controladas con el manejador de transacciones de la base de datos (*TransactionManager*).

Cuando queramos abrir iniciar una transacción (*Transaction*) usaremos el código de la Ilustración 2.

```
Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
Database dwg = Application.DocumentManager.MdiActiveDocument.Database;
using (Transaction tr = dwg.TransactionManager.StartTransaction())
{
    try
    {
        //To Do Code
        tr.Commit();
    }
    catch (System.Exception exc)
    {
        ed.WriteMessage(exc.Message);
        tr.Abort();
    }
}
```

Ilustración 2 Snippet de código para iniciar una transacción

El editor lo usaremos para imprimir en la consola de AutoCAD los errores encontrados.

En “To Do Code” se escribe el código para acceder en la transacción.

Los objetos de la aplicación son accedidos mediante su *ObjectId* de las siguientes maneras:

Si usamos directamente el Id (*ObjectId*) del objeto se usará el método

```
public DBObject GetObject(OpenMode mode);
```

Usando el objeto transacción (*Transaction*) se usará el método

```
public virtual DBObject GetObject(ObjectId id, OpenMode mode);
```

Los objetos se abren en modo lectura (*OpenMode.ForRead*) o modo escritura (*OpenMode.ForWrite*) para cambiar un objeto a modo escritura se usa el método *UpgradeUpon*.

Dibujando una línea en AutoCAD

Ejercicio 1 – Dibujado de una casita en AutoCAD

Realizar el dibujado de la siguiente figura.

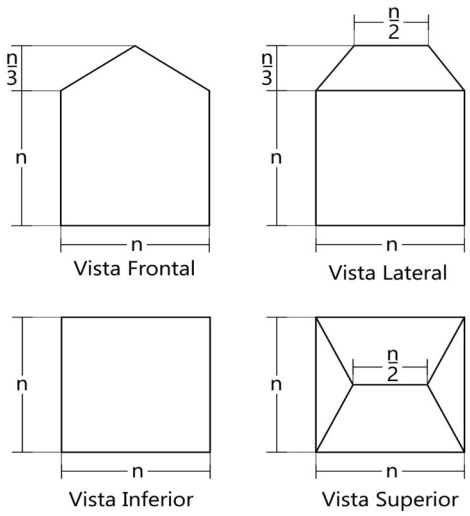
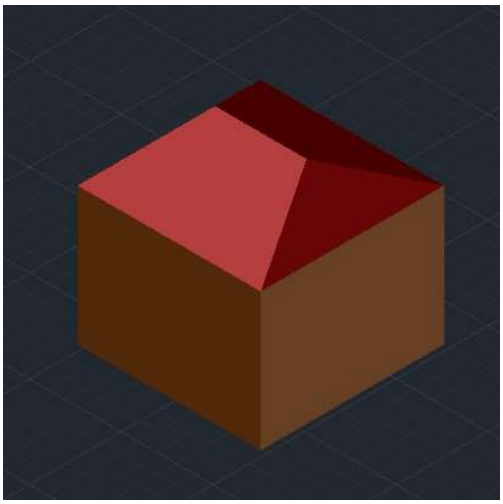


Ilustración 5 Planos de la casita

La vista de la figura en 3D es



Instrucciones

1. Crear una clase llamada *Casita*, la clase será de tipo de instancia.
2. La clase debe implementar las siguientes variables

```
/// <summary>
/// Aquí se guardan las caras que forman la geometría
/// </summary>
public List<Face> Faces;
/// <summary>
/// Aquí se guarda el punto de inserción
/// </summary>
public Point3d InsPt;
/// <summary>
/// El tamaño de la casita
/// </summary>
public double Size;
```

Ilustración 6 Propiedades de la casita

3. Un constructor que reciba el punto de inserción de la geometría, el tamaño de la casita y se encargue de llenar la lista de Faces con las caras que definen a la casa.

```
/// <summary>
/// El constructor para crear una casita, especificando un tamaño
/// y el punto de inserción
/// </summary>
/// <param name="size">El tamaño de la casita</param>
/// <param name="insPt">El punto de inserción de la casita</param>
0 references
public Casita(double size, Point3d insPt)
{
    //TO DO CODE
}
```

Ilustración 7 Constructor de la casita

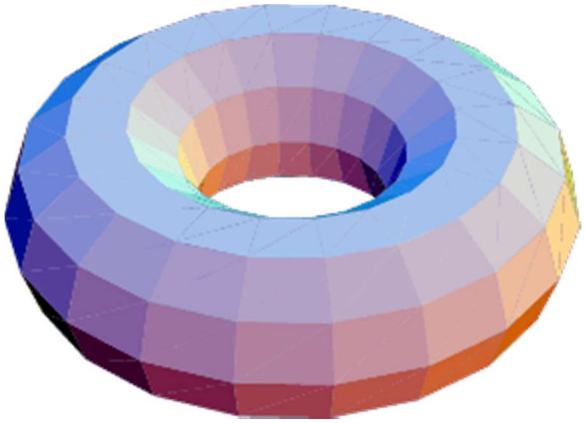
Nota. No olvidar inicializar la lista Faces antes de llenarla

4. Usar un color para el techo y otro para las paredes.
5. En la clase de *Commands* crear un comando que dibuje la casita seleccionando dos puntos, el tamaño de la casita se define como la distancia entre los dos puntos. El nombre del comando será "**NCasita**".

Nota. Los colores para la casita pueden ser otros a los mostrados en la imagen

Ejercicio 2 – Toroide

Dibujado de un toroide, con cuadrados.



Instrucciones

1. El toroide se define con las siguientes ecuaciones paramétricas

$$x = (c + a\cos v)\cos u$$
$$y = (c + a\cos v)\sin u$$
$$z = a\sin v$$

2. El muestreo de cuadrados será definido por las siguientes variables.
   - **c: La distancia del centro del tubo al centro del toroide, podemos considerarlo como el radio 1.**

- **a: Es el radio que define el grosor del toroide.**
- **n: será el número de veces que se divide el ángulo u**
- **m: será el número de veces que se divide el ángulo v**

Una propuesta del constructor es:

```
public Toroide(Double _R, Double _r, int _u, int _v, Point3d center)
```

3. Cada cuadrado del Toroide se dibujará con un color aleatorio.
4. Con base a los puntos anteriores crear una clase que se llame *Toroide* de tipo instancia.
5. La clase debe implementar las siguientes variables

```
/// <summary>
// La distancia del centro del tubo al centro del toroide
/// </summary>
public double R;
/// <summary>
/// El radio del tubo
/// </summary>
public double r;
/// <summary>
/// El número con el que se divide θ
/// </summary>
public int U;
/// <summary>
/// El número con el que se divide ϕ
/// </summary>
public int V;
/// <summary>
/// El centro para el toroide
/// </summary>
public Point3d Center;
/// <summary>
/// La geometría del toroide
/// </summary>
public Point3dCollection Geometry;
/// <summary>
/// La lista de caras a dibujar
/// </summary>
public List<Face> Faces;
```

Ilustración 8 Propiedades del toroide

6. Una función *Draw*, que llene la colección de *Geometry* y utilice el centro solicitado al usuario para dibujar el toroide.

Nota: Todas las caras deben ser dibujadas en una sola transacción.

Ejercicio 3 – El triángulo de Sierpiński

Métodos auxiliares

Con estas funciones pueden apoyarse para resolver la tarea.

Dibuja un grupo de entidades

```
public static ObjectIdCollection Draw(params Entity[] ents)
{
    ObjectIdCollection ids = new ObjectIdCollection();
    Editor ed = Application.DocumentManager.MdiActiveDocument.Editor;
    Database dwg = Application.DocumentManager.MdiActiveDocument.Database;
    using (Transaction tr = dwg.TransactionManager.StartTransaction())
    {
        try
        {
            //Abre el espacio activo
            BlockTableRecord currenSpace = (BlockTableRecord)
                dwg.CurrentSpaceId.GetObject(OpenMode.ForWrite);
            foreach (Entity ent in ents)
            {
                currenSpace.AppendEntity(ent);
                tr.AddNewlyCreatedDBObject(ent, true);
                ids.Add(ent.Id);
            }
            tr.Commit();
        }
        catch (Exception exc)
        {
            ed.WriteMessage(exc.Message);
            tr.Abort();
        }
    }
    return ids;
}
```

Convierte un punto 3d a 2d

```
        public static Point2d ConvertTo2d(Point3d point3d)
        {
            return new Point2d(point3d.X, point3d.Y);
        }
```

Entregables

Las respuestas al cuestionario de la práctica en un documento, PDF, formato preguntas y respuestas.

En el cuestionario agregar nombre completo.

Agregar las clases

- *Casita.cs*
- *Toroide.cs*
- *Sierpinski.cs*
- *Commands.cs*
- La imagen de la casita en una perspectiva 3D
- La imagen del toroide en una perspectiva 3D
- La imagen del triángulo n=6, con el total de triángulos dibujados anexado al cuestionario.

Solo estos archivos comprimidos en un RAR y enviados a mi correo miguel.alanis@dasoft.com.mx

En caso de haber problemas al mandar por correo compartir el link de una carpeta de la nube.