3. PROGRAMACIÓN BÁSICA EN AUTOCAD

¿Qué es un dibujo de AutoCAD?
¿Qué es el API ObjectARX?
¿Cómo se puede crear y manipular dibujos desde programación?

Autor: Jorge A. Díez Pomares
Laboratorio de C.A.D. UMH

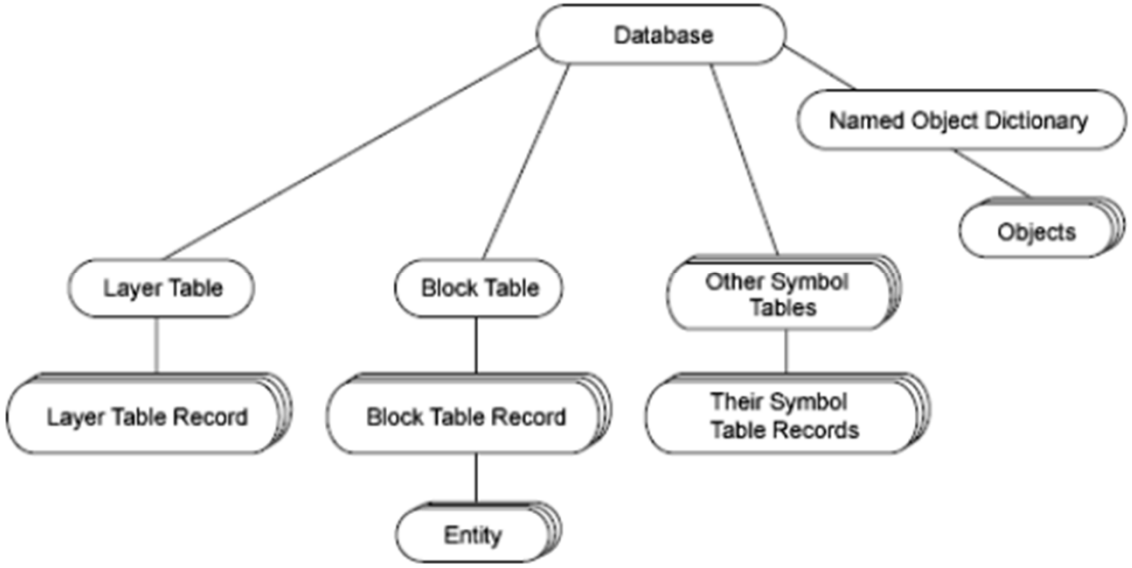
Índice:

3.1. CONCEPTOS BÁSICOS

- Diseño de programas que se ejecutarán en el entorno AutoCAD
- Creación de nuevos comandos
- Uso de API de desarrollo de Autodesk para manipular bases de datos .dwg

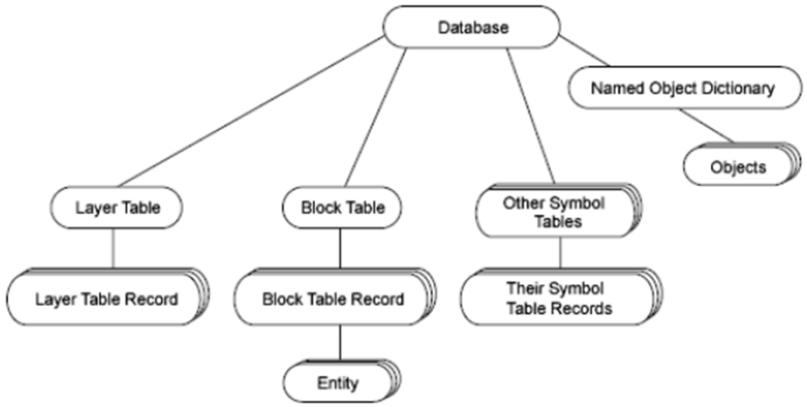
3.1.1. DIBUJO .DWG

- Un dibujo de AutoCAD es una colección de objetos almacenados en una base de datos.
- Contiene “database objects” (DBObjects)
- DBObjects básicos: “Dictionaries”, “Symbol tables”, “Entities”



3.1.1. DIBUJO .DWG

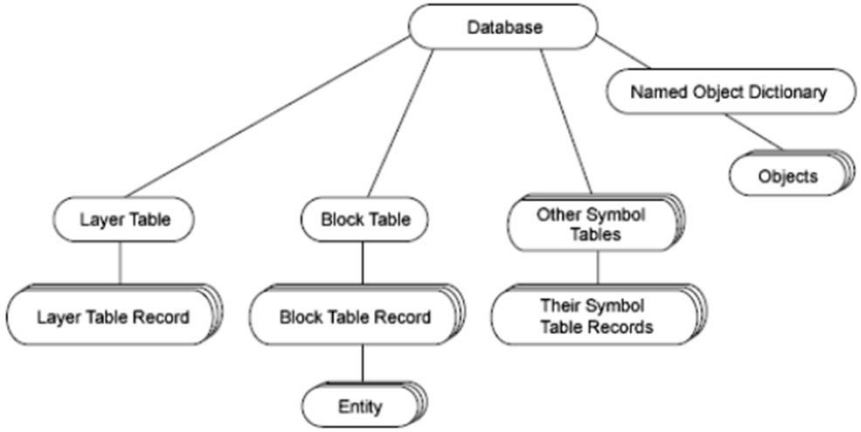
- “Symbol tables” y “Dictionaries” son contenedores de otros DBObjects.
- “Symbol tables”: Número fijo (no se pueden añadir), almacenan instancias de una clase asociada a cada tabla (Symbol table record)



Symbol Tables
BlockTable
DimStyleTable
LayerTable
LinetypeTable
RegAppTable
TextStyleTable
UcsTable
ViewportTable
ViewTable

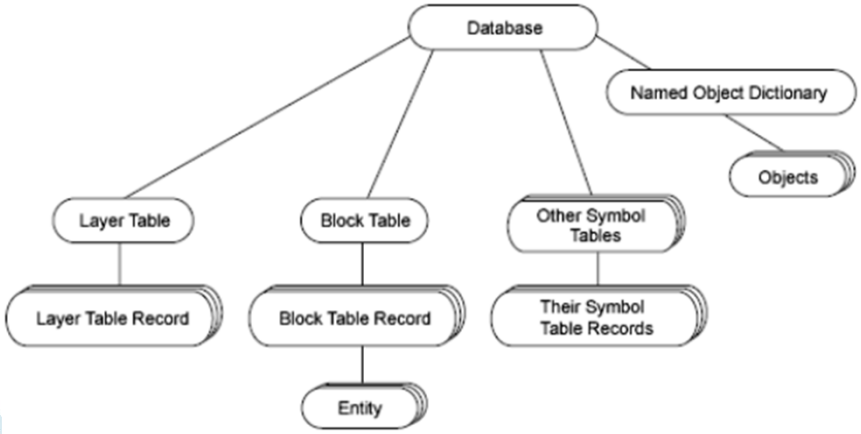
3.1.1. DIBUJO .DWG

- Entidades (“Entities”), son una clase particular de DBObject con representación gráfica.
- Líneas, Círculos, Splines, Cotas, Texto, Bloques, Sólidos…
- Tienen que estar contenidos en un “Block Table Record” de la Block Table.
- Los Block Table Record se corresponden con el espacio modelo y las distintas presentaciones de espacio papel.



3.1.1. DIBUJO .DWG

- Los diccionarios (Dictionaries) son contenedores genéricos. Un diccionario puede almacenar cualquier objeto tipo DBObject y derivados
- Al crear un nuevo dibujo siempre se crea un "NamedObjectDictionary" que puede contener objetos y diccionarios, definidos por el programador.
- Permiten extender la base de datos del dibujo para almacenar datos definidos por nosotros.



3.1.1. DIBUJO .DWG

- En una misma sesión de AutoCAD se puede manejar varias bases de datos a la vez.
- Identificadores de objetos:
  - Manejador (Handle): Identifica de manera única al objeto dentro de la propia base de datos. El valor de este manejador se guarda en la propia base de datos, por lo que el manejador se mantiene entre distintas sesiones. No se garantiza que el handle sea único entre distintas bases de datos.
  - ID de objeto (ObjectID): Identifica únicamente al objeto entre distintas bases de datos abiertas a la vez. Sin embargo el objectID no persiste entre sesiones, variará cada vez que cerremos o abramos el contenedor del objeto.
- Requisitos básicos de la base de datos para ser usable:
  - Debe contener los 9 Symbol Table
  - El Block Table debe contener tres Block Table Record : MODEL_SPACE, PAPER_SPACE, PAPER_SPACE0
  - El Layer Table debe contener un Layer Record correspondiente con la capa 0
  - El LineType Table debe contener el tipo de línea CONTINUOUS
  - Debe contener un NamedObjectDictinary con 4 diccionarios: GROUP, MLINE STYLE, LAYOUT, PLOT STYLE. En el "MLINE style dictionary", debe existir siempre el estilo STANDARD.
- La función encargada de crear bases de datos desde cero inicializará estos componentes básicos si así se le indica.

3.1.2. API OBJECT ARX

- API: Application programming interface
- Conjunto de clases, funciones, estructura… en forma de biblioteca, que añade una capa de abstracción a la programación.
- Las funciones implementadas permiten realizar tareas de cierta complejidad sin tener que conocer con detalle las capas de menor abstracción.
- Así, la API ObjectARX permite crear y manipular bases de datos y objetos de AutoCAD sin necesidad de un conocimiento profundo de como realiza AutoCAD estas tareas a nivel interno.
- Funciones programadas en C++

3.1.3. ".NET" MANAGED WRAPPER CLASSES

- La API ObjectARX está programada en C++, pero incluye librerías que "envuelven" (wrap) estas funciones en forma de funciones para lenguajes .NET
- Las llamadas que realicemos a funciones en C#, se traducirán en llamadas a funciones C++ que realizarán su correspondiente tarea y devolverán las salidas necesarias.
- Los wrappers están contenidos en tres ensamblajes:
  - **acdbmgd.dll** contiene los wrapper relacionados con la gestión de la base de datos .dwg
  - **acdbmgdbrep.dll** incluye los wrapper relacionados con la representación de contornos para sólidos y superficies. (No es objeto del curso)
  - **acmgd.dll** cotiene los wrapper del resto de funciones y clases de la API.
- Las bibliotecas acdbmgd y acmgd serán indispensables para la mayoría de las aplicaciones sobre AutoCAD.

3.2. CREACIÓN DE COMANDOS PERSONALIZADOS

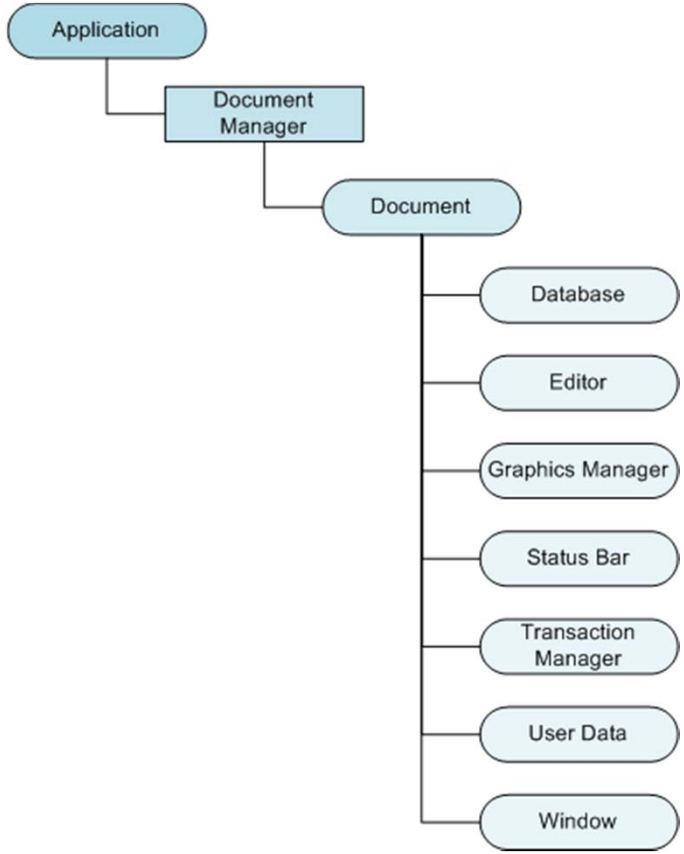
- La programación permite modificar completamente el comportamiento de AutoCAD (Ej: Civil 3D, Architectural Desktop, AutoCAD Mechanical…):
  - Definir variables de entorno nuevas
  - Modificar la interfaz gráfica de usuario
  - Nuevas entidades con representación gráfica distinta
  - Nuevos comandos
- La funcionalidad del programa sin embargo viene dada por los Comandos, que son los que permiten una interacción entre el sistema y el usuario (Convertir las entradas en salidas).

3.2. CREACIÓN DE COMANDOS PERSONALIZADOS

- Para definir un comando, se requiere un método público y estático que no requiera parámetros y que tenga no devuelva ninguna salida. Es decir, un método del tipo:
  - *public static void metodoComando (){}*
- Las entradas y salidas serán gestionadas dentro de la propia función mediante peticiones al usuario (selección de entidades, entrada por línea de comandos, ventanas de diálogo, etc)
- Declaración de un método como comando:
  - *[CommandMethod("HELICEESF", CommandFlags)]*
    public static void Funcion() {//...}
- El parámetro CommandFlags permite establecer el comportamiento del comando (si es transparente, modal, si se puede ejecutar en espacio papel, etc.), es un parámetro opcional

3.3. MANIPULACIÓN DE DIBUJOS .DWG: CREACIÓN, APERTURA Y GUARDADO

- Application: Objeto que contiene información sobre la aplicación AutoCAD abierta.
- El “Document manager” administra todos los dibujos abiertos en esa aplicación.
- Cada dibujo está asociado a un objeto de clase Document.
- La clase Document contiene la base de datos del dibujo (objeto de clase Database) y los elementos necesarios para la interacción con la base de datos.



3.3. MANIPULACIÓN DE DIBUJOS .DWG: CREACIÓN, APERTURA Y GUARDADO

- Obtención del documento actual:
  - *Document docActual=Application.DocumentManager.MdiActiveDocument;*
- Creación de un documento nuevo:
  1. Se obtiene una referencia al Document Manager
  2. Se añade una instancia con la plantilla deseada al Document Manager, y este nos devolverá un objeto de tipo Document listo para usar.
  3. (Opcional) Si lo deseamos podemos establecer el nuevo documento como documento activo igualando el ActiveDocument del Document Manager al Document obtenido.

Ejemplo:

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("Nuevo dibujo", CommandFlags.Session)]
public static void NuevoDibujo(){
string rutaPlantilla = "acad.dwt";

DocumentCollection acDocMgr = Application.DocumentManager;// Paso 1
Document acDoc = acDocMgr.Add(rutaPlantilla); //Paso 2
acDocMgr.MdiActiveDocument = acDoc; //Paso 3
}
```

3.3. MANIPULACIÓN DE DIBUJOS .DWG: CREACIÓN, APERTURA Y GUARDADO

- Para abrir un dibujo ya existente en un documento nuevo se utiliza:

```
using System.IO;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;

[CommandMethod("AbrirDibujo", CommandFlags.Session)]

public static void AbreDibujo() {
string rutaDibujo = "C:\\campus.dwg";  //Ruta a cargar
DocumentCollection acDocMgr = Application.DocumentManager;  //Obtención de DM
if (File.Exists(rutaDibujo)) {  //Comprobamos si existe el fichero
Document docNuevo = acDocMgr.Open(strFileName, false); //Instrucción de apertura
 }
else {
acDocMgr.MdiActiveDocument.Editor.WriteMessage("File "+ rutaDibujo + " does not exist.");
 }
}
```

3.3. MANIPULACIÓN DE DIBUJOS .DWG: CREACIÓN, APERTURA Y GUARDADO

- El guardado del .dwg se realiza a nivel del objeto Database, no del objeto Document, por lo que se llama desde el objeto Database asociado al Document:
  - *documento.Database.SaveAs(parametros)*: *Consultad parámetros en la referencia*
- Cierre del dibujo:
  - *documento.CloseAndSave(ruta);*
  - *documento.CloseAndDiscard();*

3.3. MANIPULACIÓN DE DIBUJOS .DWG: CREACIÓN, APERTURA Y GUARDADO

- Es posible crear, modificar y guardar bases de datos que no estén asociadas a ningún documento. Pero no será posible la interacción directa con el usuario.
- Creación:
  - *Database db=new Database(porDefecto, noDocument);*
  - *db.ReadDWG(ruta, FileOpenMode, allowCPconversion, password);*
- Guardado:
  - *db.SaveAs(parámetros);*
- Cierre:
  - *db.Close();*

3.4. BASE DE DATOS DE AUTOCAD: TRANSACCIONES

- Base de datos de AutoCAD es Transaccional.
- **Transaccional**: La base de datos se modifica entre estados consistentes mediante transacciones.
- **Transacción**: interacción con una estructura de datos compleja, compuesta por varios procesos que se han de aplicar uno después del otro. La transacción debe realizarse de una sola vez y sin que la estructura a medio manipular pueda ser alcanzada por el resto del sistema hasta que se hayan finalizado todos sus procesos.
- Las instrucciones dentro de la transacción se ejecutan de forma atómica.

3.4. BASE DE DATOS DE AUTOCAD: TRANSACCIONES

- La lectura o adición de datos a la base de datos .dwg se realiza mediante transacciones.
- La transacción se inicia a partir del atributo Transaction Manager de la base de datos.
  - *Transaction tr=db.TransactionManager.StartTransaction();*
- El código implementado dentro de la transacción se suele colocar dentro de un bloque "using". Este bloque define el dominio de una variable (no existirá fuera de el). De esta forma, la transacción creada se destruye automáticamente al final del bloque.

```
Using (Transacion tr=db.TransactionManager.StartTransaction()){
        (bloque de código que utiliza la variable tr)
} //A partir de este punto tr se habrá eliminado de memoria
```

3.4. BASE DE DATOS DE AUTOCAD: TRANSACCIONES

- Para que los cambios realizados sean efectivos se debe realizar un “commit”
- Sin embargo, si durante la transacción de ha producido algún error, se puede devolver la base de datos al estado consistente anterior utilizando un “rollback” (Abort)

Ejemplo:

```
Using (Transacion tr=db.TransactionManager.StartTransaction()){
        try{
                (operaciones)
        }
        catch{
                tr.Abort(); //Si se produce algún error se vuelve al estado anterior
        }
        tr.Commit(); //Si no, se guarda el nuevo estado como consistente
}
```

3.4. BASE DE DATOS DE AUTOCAD: TRANSACCIONES

- Dentro de la transacción es habitual acceder a los Symbol Table (BlockTable para entidades, LayerTable para modificar capas, etc.)
- Se utiliza el método de la transacción (devuelve un objeto de tipo
  - *(Casting)tr.GetObject(objectID,modoapertura);*
- Obtención de ID para los Symbol Table de la base de datos “db”
  - *db.BlockTableID, db.LayerTableID, ect.*

3.5. ENTIDADES

- Objetos de la base de datos con representación gráfica (Líneas, Arcos, Texto, Cotas, Superficies, Bloques…)
- Tienen que estar dentro de un BlockTableRecord (Espacio modelo, presentaciones)
- Las distintas clases que representan los distintos tipos de entidad están estructuradas en un árbol de herencias.
  - La superclase Entity (que es subclase de DBObject) tiene multitud de subclases derivadas (31 subclases) de las cuales derivan otras tantas subclases.
  - Consultad en la documentación de la API ObjectARX

3.5. ENTIDADES

- Las entidades se crean a partir del constructor
- Sólo las subclases que representan una entidad concreta tienen constructor (Ej: La clase Curve no tiene constructor, pero la clase Line que deriva de ella sí)
- Para dibujarlas es necesario añadirlas a la base de datos.
  1. Se comienza una transacción en la base de datos de destino
  2. Se obtiene el BlockTable de la base de datos
  3. Se obtiene el BlockTableRecord donde dibujar
  4. Se añade la entidad al BlockTableRecord
  5. Se asignan las propiedades básicas que requiere la entidad (capa, tipo de linea, etc.)
  6. Se cierra la transacción

3.5. ENTIDADES

Ejemplo:

```
public static void dibujaListaNuevos(List<Entity> lista,Database dbdestino){

        //Abrimos una transaccion para añadir las entidades a la base de datos
        Transaction tr = dbdestino.TransactionManager.StartTransaction();

        //Comenzamos la transaccion
        using (tr)
        {
            //Obtenemos el blocktable y el blocktablerecord
            BlockTable bt = (BlockTable)tr.GetObject(dbdestino.BlockTableId, OpenMode.ForRead);
            BlockTableRecord btr = (BlockTableRecord)tr.GetObject(bt[BlockTableRecord.ModelSpace], OpenMode.ForWrite);

            //Añade cada objeto a la base de datos
            foreach (Entity ent in lista)
            {
                btr.AppendEntity(ent); //Añadimos la entidad al blocktablerecord

                tr.AddNewlyCreatedDBObject(ent, true); //Asigna las propiedades necesarias

            }
            //Cerramos la transacción
            tr.Commit();
        }

    }
```

3.5. ENTIDADES

- Para copiar entidades entre bases de datos se utiliza la función:
  - Origendb.WblockCloneObjects(parametros);

Ejemplo:

```
public void dibujarLista(List<Entity> lista, Database destinodb, Database origendb)
        {
            //Comenzamos la transaccion
            using (Transaction tr = destinodb.TransactionManager.StartTransaction())
            {
            ObjectIdCollection idslista=new ObjectIdCollection();
            //Obtenemos las objectId de los objetos
            foreach (Entity entidad in lista)
            {
                idslista.Add(entidad.ObjectId);
            }
            //Abrimos el blockTable
            BlockTable destinobt = (BlockTable)tr.GetObject(destinodb.BlockTableId, OpenMode.ForRead);
            //Obtenemos el blockTableRecord del espacio modelo
            BlockTableRecord destinobtr = (BlockTableRecord)tr.GetObject(destinobt[BlockTableRecord.ModelSpace], OpenMode.ForWrite);
            //Creamos un idmaping
            IdMapping idmap=new IdMapping();

            //Con la lsita de ids, copiamos las entidades entre bases de datos
            origendb.WblockCloneObjects(idslista, destinobtr.ObjectId, idmap, DuplicateRecordCloning.Ignore, false);
            tr.Commit();
        }
        }
```

3.6. MANEJO DE DOCUMENTACIÓN

- En las versiones actuales de los ObjectARX, la documentación se descarga por separado.
- Se puede integrar en Visual Studio
- La documentación se encuentra en la carpeta "doc"
- La referencia de las bibliotecas .NET corresponde al "arxmgd.chm"
- Más ayuda en Autodesk Developer Network (ADN)
  - Foros
  - Tutoriales
  - Guías