

2

(XII)

Prototype

(Patrones de diseño)

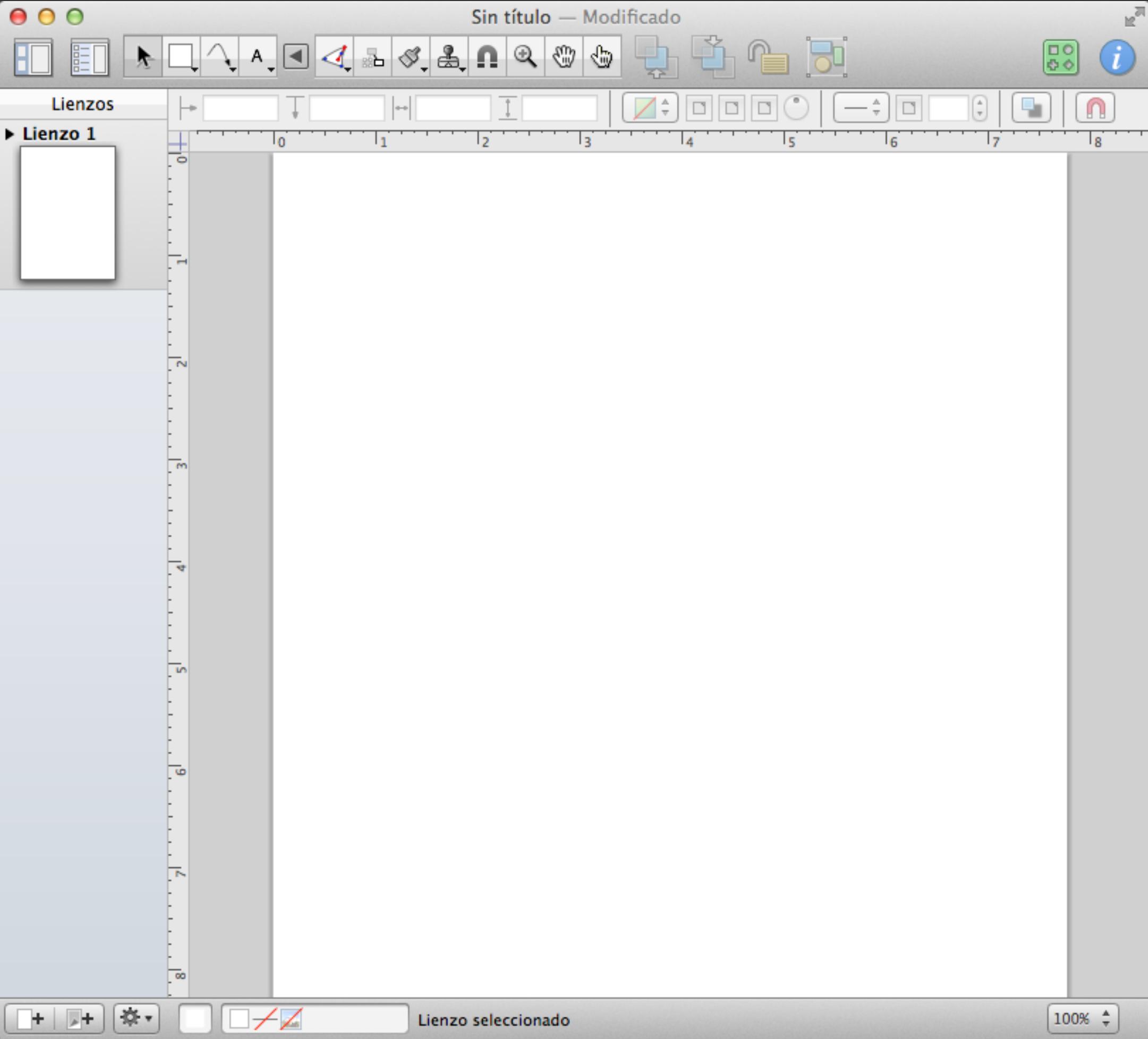
Diseño del Software

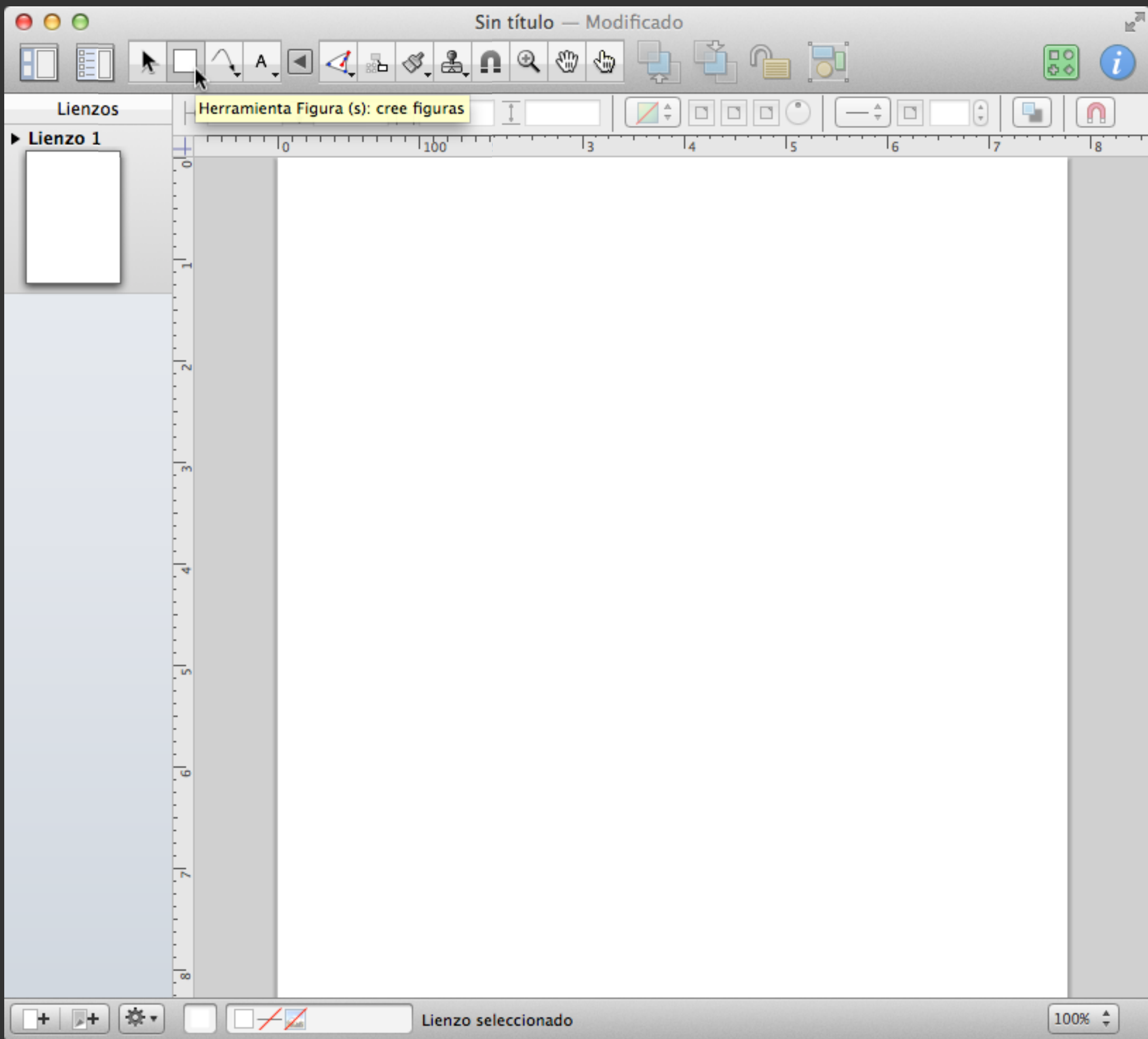
Grado en Ingeniería Informática del Software

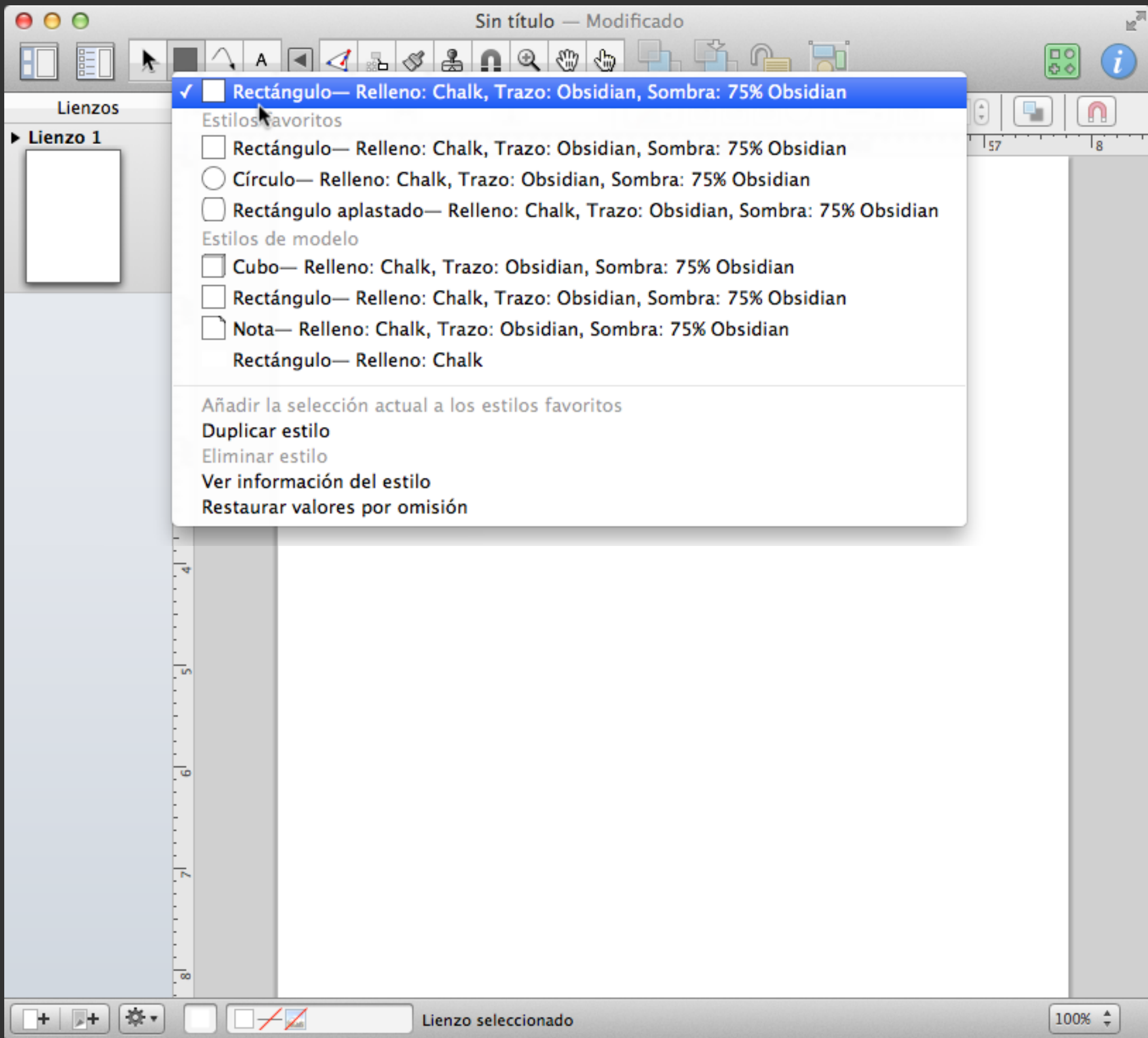
Curso 2017-2018

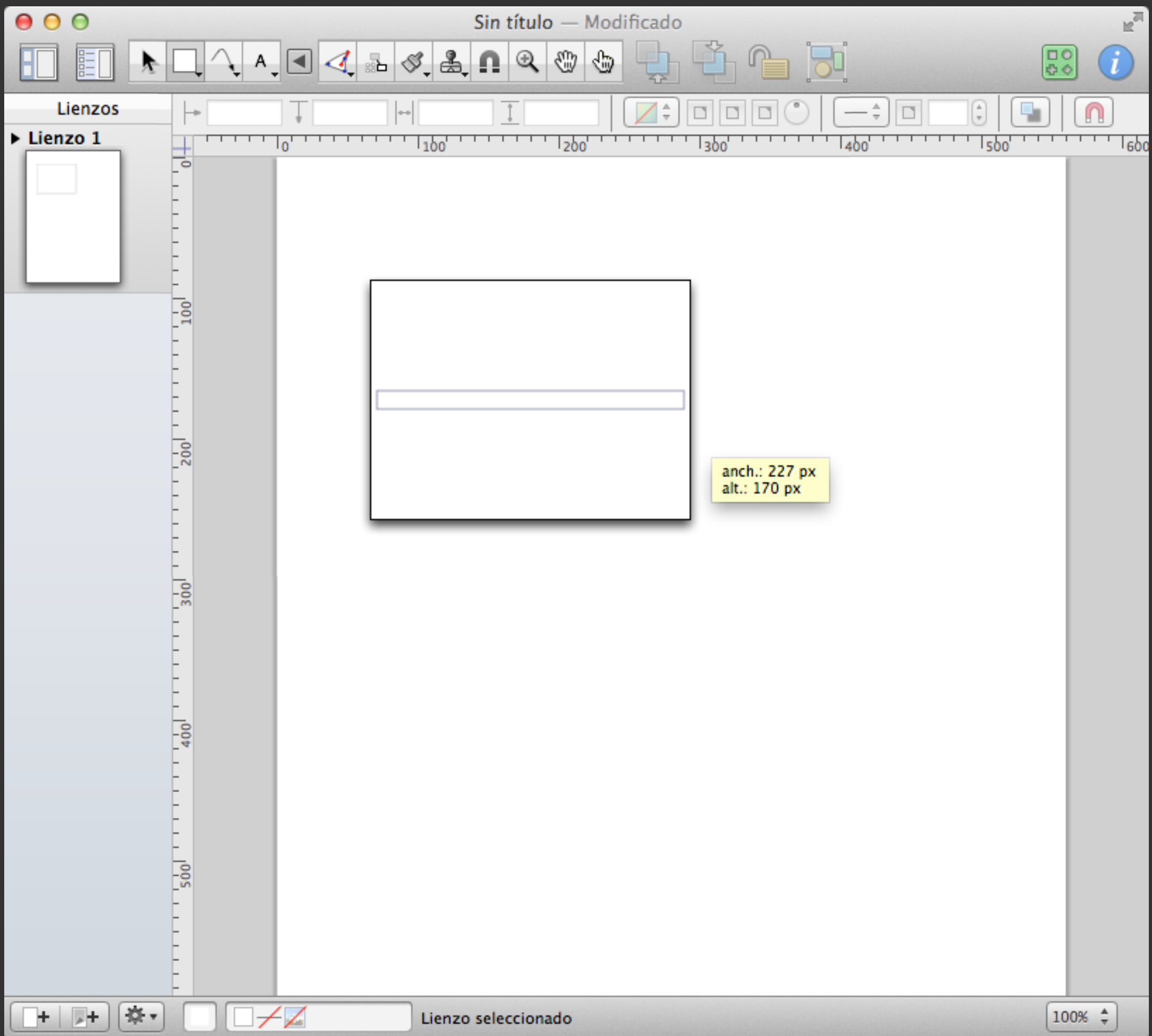
Ejemplo

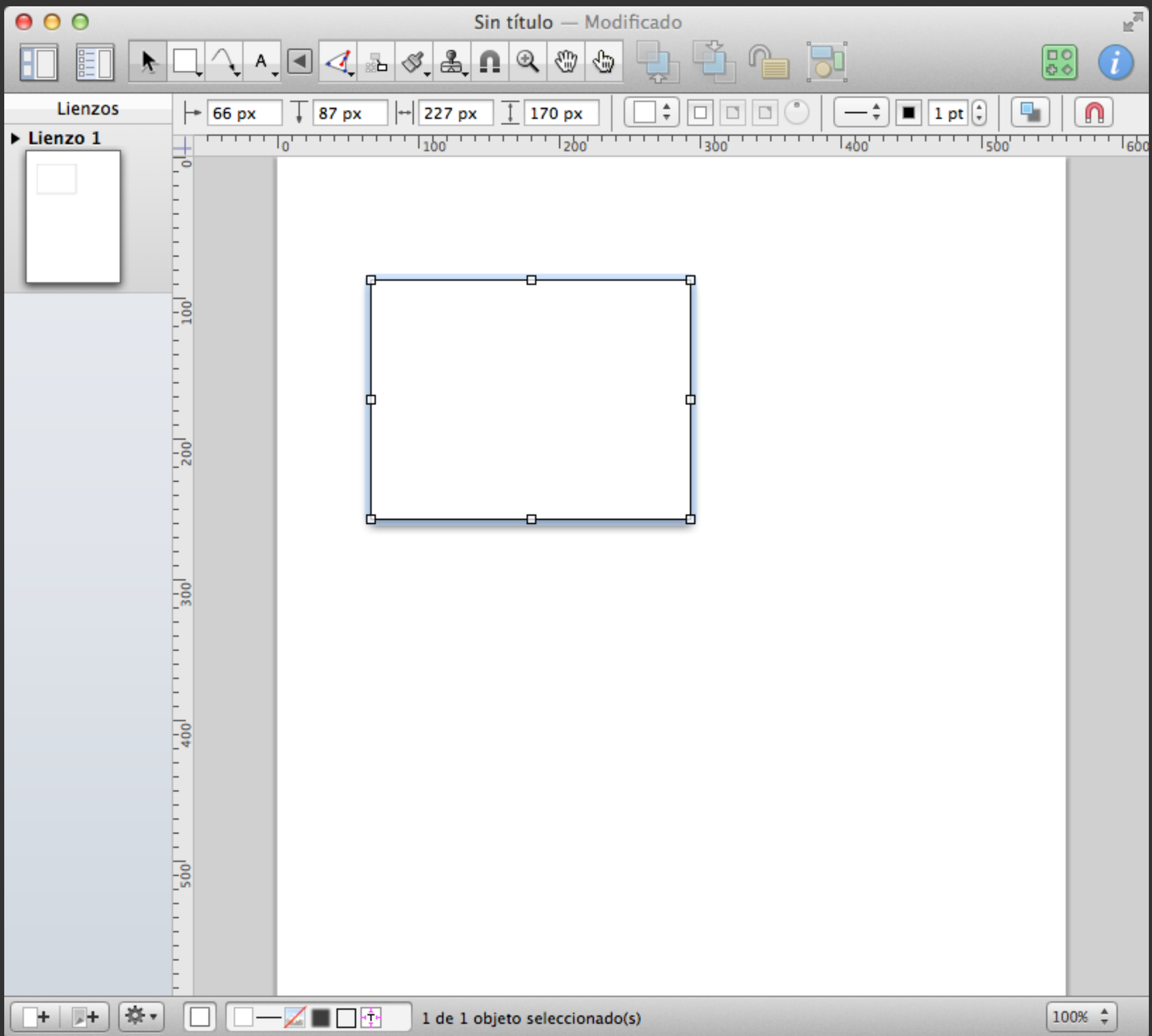
Veamos un posible ejemplo de aplicación del patrón tomando como ejemplo una herramienta de creación de diagramas (concretamente, OmniGraffle, para Mac).









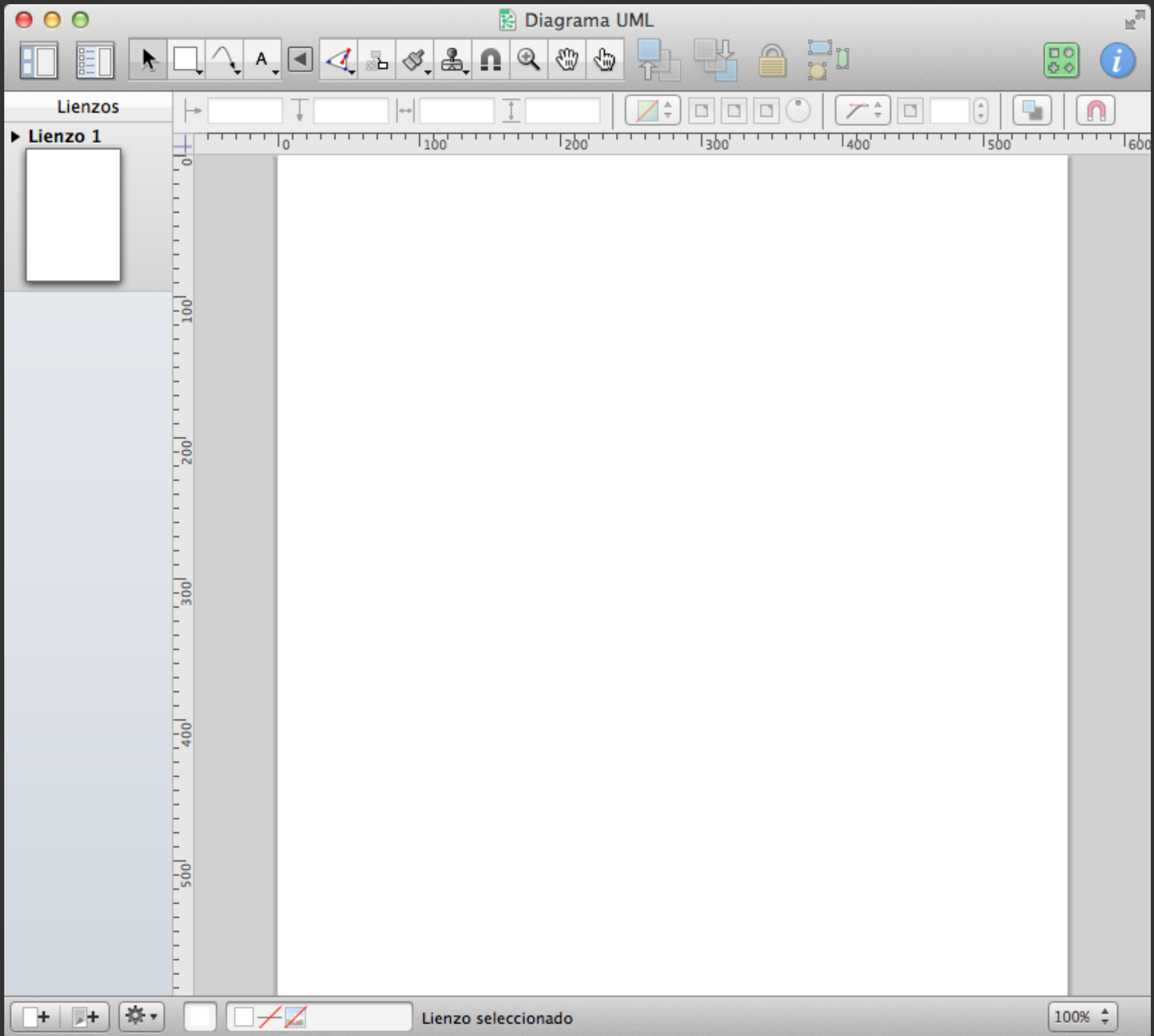


¿Qué hemos hecho?

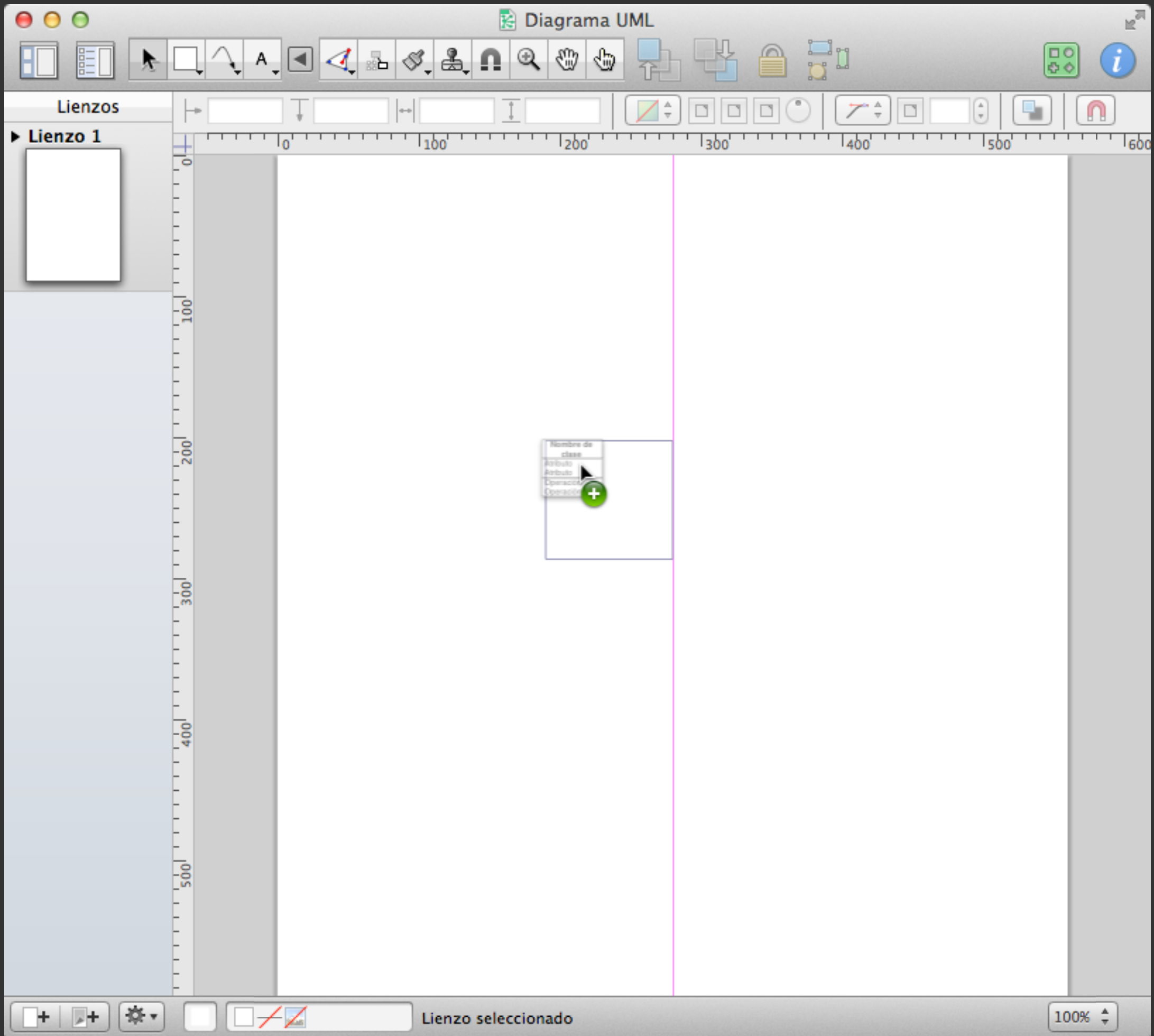
- **Hasta aquí, nada que no supiéramos cómo diseñar**
 - Hemos creado uno de los tipos de figuras predefinidos (en este caso, un rectángulo)
 - Muy probablemente, cada uno de ellos se modelarían como clases distintas
 - ▶ No es lo mismo dibujar un rectángulo que un círculo, un cubo, una línea, un corazón o cualquier otro símbolo predefinido

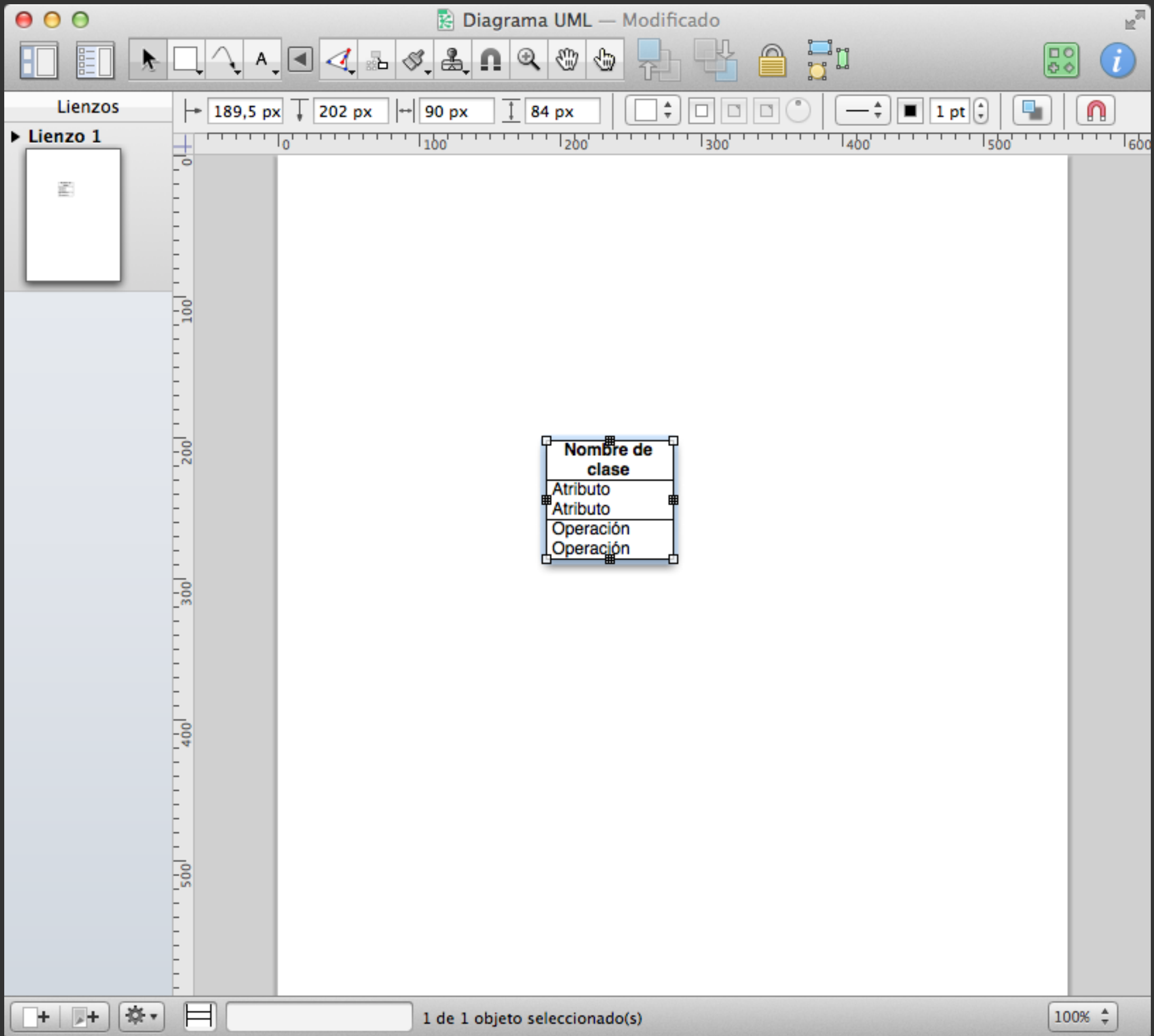
«Stencils»

- **Pero resulta que el programa debe permitir crear nuevos tipos de figuras al usuario**
- **Por ejemplo:**
 - Elementos de diagramas UML
 - «Widgets» para elaborar bocetos de interfaces gráficas de usuario
 - Cualquier otra figura compleja que deseemos guardar como plantilla para poder crear copias de ella en un futuro









Problema

- **El programa, obviamente, no puede conocer a priori dichos tipos de figura**
 - Son infinitos
 - Se crean dinámicamente

Solución

- El patrón *Prototype* (prototipo)
- Básicamente, consiste en que los objetos sepan cómo clonarse a sí mismos

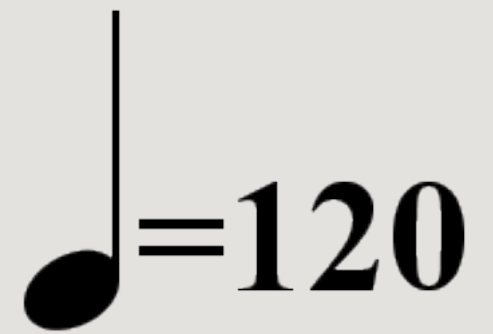
Prototype (Prototipo)

- Patrón de creación (ámbito de objetos)
- Propósito:

Especifica los tipos de objetos a crear usando una instancia prototípica, y crea nuevos objetos copiando dicho prototipo.

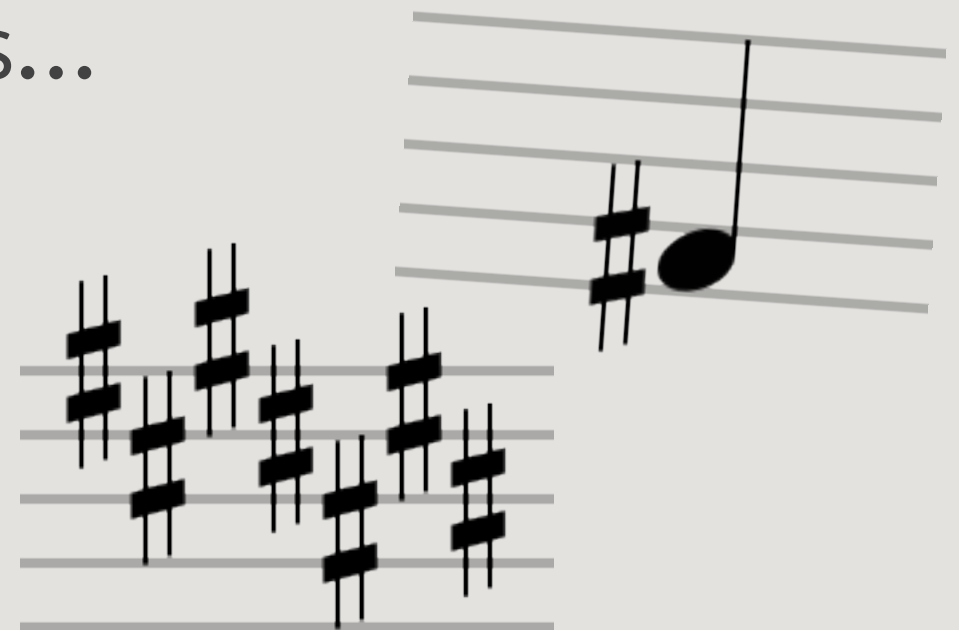


Motivación



veamos ahora el ejemplo que da el GoF en la sección de motivación (similar al nuestro)

- **Tenemos que crear un framework para editores gráficos**
- **Un usuario podría construir con él un editor de partituras musicales**
 - Permitiría añadir los distintos tipos de notas, silencios, claves, ligaduras...



Motivación

- **Supongamos que el framework provee una clase abstracta `Graphic` para los elementos gráficos**
 - Nuestro editor musical definirá las distintas clases necesarias para sus necesidades
- **El framework también proporciona una clase `GraphicTool` para los elementos de la paleta que permiten crear símbolos gráficos**

Motivación

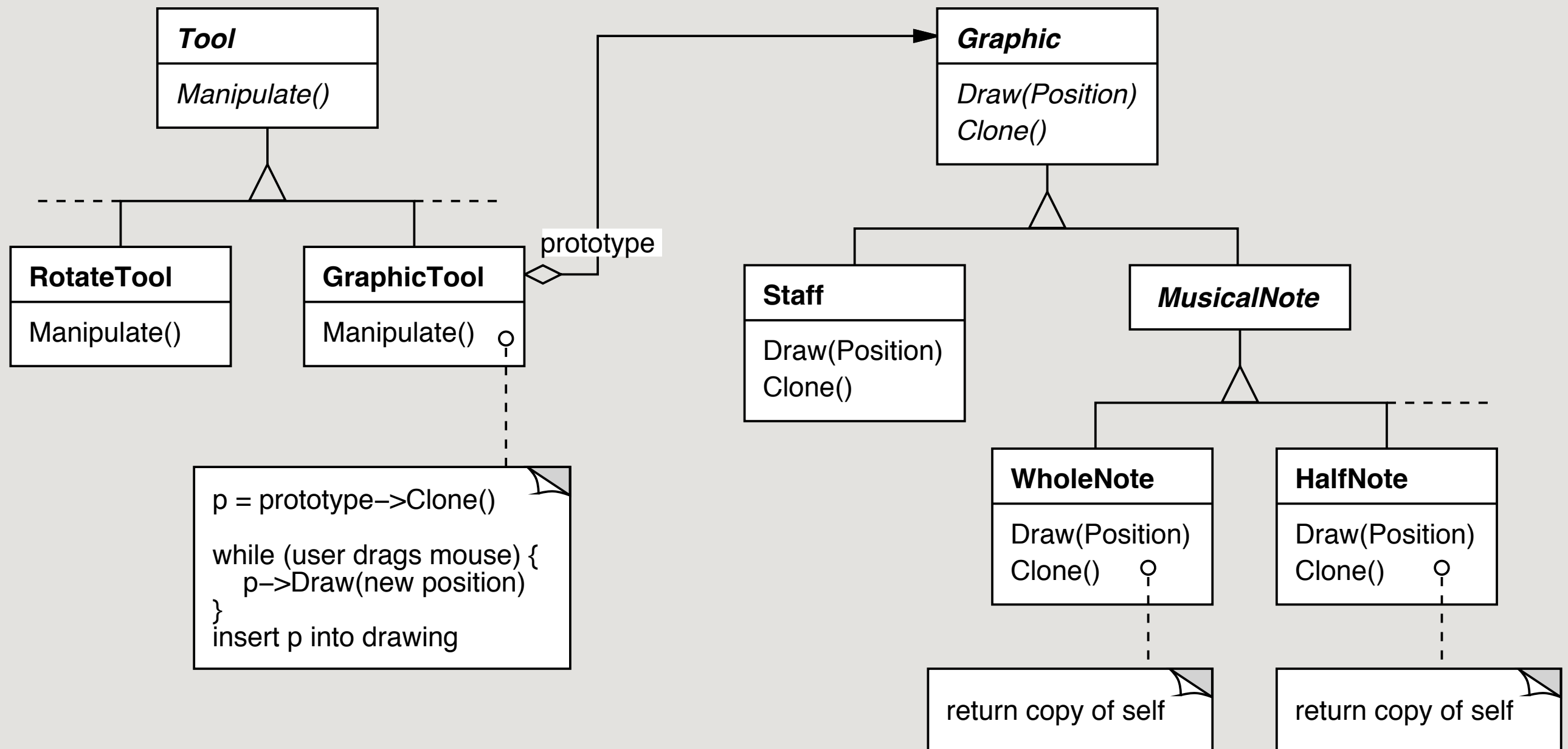
- **Una opción sería que el editor musical definiese tantos tipos de subclases de `GraphicTool` como posibles figuras**
 - Por cierto, ¿qué patrón sería ése?
- **Pero, en este caso, esas subclases serían totalmente artificiales**
 - Sólo se diferenciarían en el tipo de símbolo musical a crear

Además, esa solución no valdría para nuestro ejemplo: ahí los nuevos tipos de símbolos gráficos son creados dinámicamente, no por otro programador extendiendo un framework, como este caso, sino... ¡por el propio usuario!

Motivación

- **¿Cómo podríamos parametrizar `GraphicTool` con el tipo de objeto a crear, aplicando la composición de objetos?**
- **Solución:**
 - Haciendo que cada instancia de ella reciba en el constructor un objeto representando el tipo de figura a crear...
 - Y que cada figura sepa cómo clonarse a sí misma

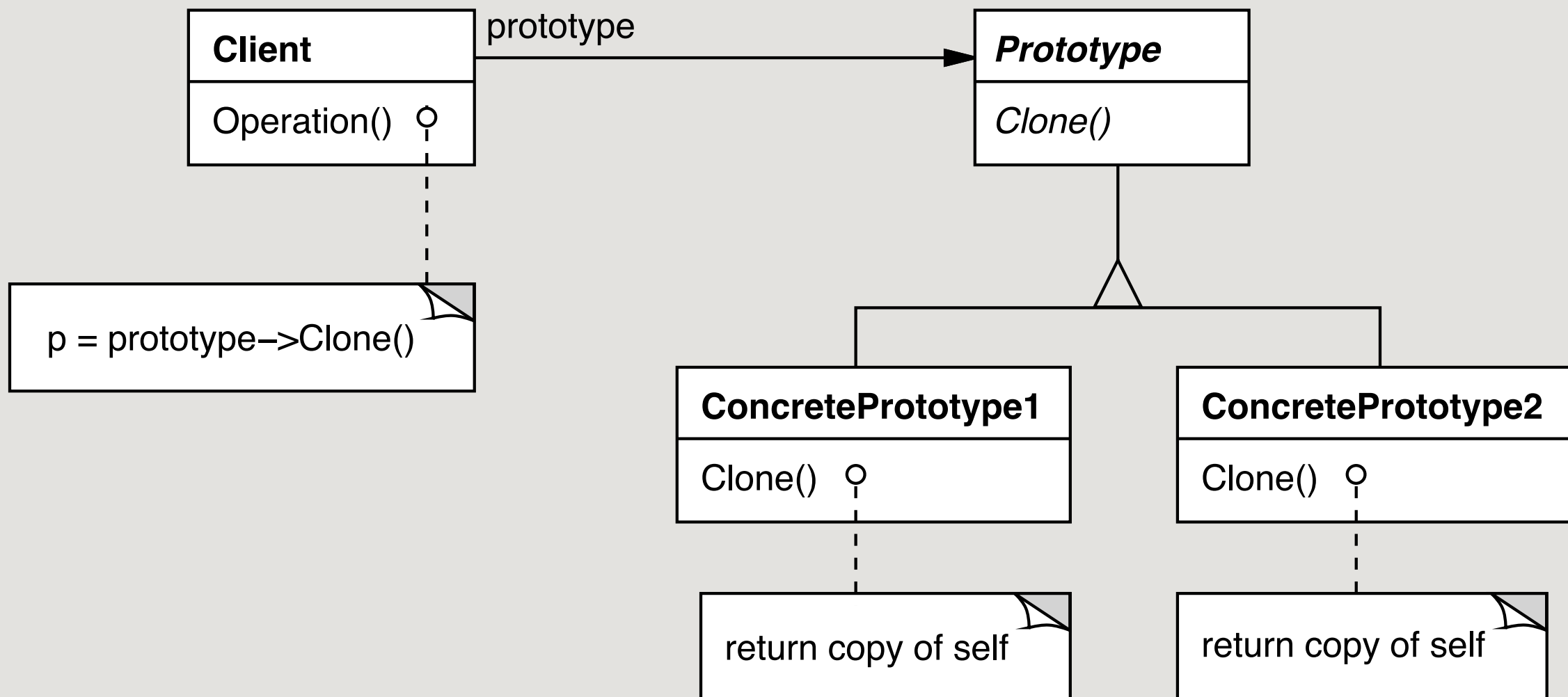
Motivación



Aplicabilidad

- © Úse el patrón *Prototype* cuando un sistema no pueda (o no deba) conocer cómo se crean, componen y representan los productos, y además se da alguna de estas circunstancias:
 - las clases a instanciar son definidas en tiempo de ejecución
 - para evitar construir una jerarquía paralela de factorías de productos
 - cuando las instancias de una clase puedan tener sólo unos pocos posibles estados, y pueda resultar más conveniente crear los objetos correspondientes como prototipos y clonarlos, en vez de instanciar manualmente la clase, cada vez con el estado necesario

Motivación



Participantes

- **Prototype (Graphic)**

- Declara la interfaz (normalmente, una única operación) para clonarse

- **ConcretePrototype (Staff, WholeNote, HalfNote)**

- Implementa la operación de clonación

- **Client (GraphicTool)**

- Crea un nuevo objeto diciéndole al prototipo que se clone

Colaboraciones

- **Un cliente le pide al prototipo que se clone**

Consecuencias

- Como el patrón *Abstract Factory* oculta las clases concretas de producto al cliente
- Además permite:
 - Añadir y eliminar productos dinámicamente (en tiempo de ejecución)
 - Simplemente registrando de algún modo el producto en el cliente
 - Especificar nuevos objetos modificando valores de sus propiedades
 - Mediante composición de objetos
 - Es como si los usuarios creasen nuevas «clases» sin programar

Consecuencias

- Especificar nuevos objetos variando su estructura
 - ▶ A partir de partes y subpartes
 - ▶ Podemos guardar esas estructuras complejas para crearlas una y otra vez *Lo que hicimos en el ejemplo de UML*
 - ▶ ¿Qué otro patrón entrará aquí en juego? *Composite*
- Reduce las subclases *(Nota: en este caso posiblemente el compuesto tenga que implementar una copia profunda o «deep copy»)*
 - ▶ A diferencia del Factory Method, no hace falta una jerarquía paralela de clases de creación

Esto es especialmente cierto en lenguajes como Java o C++, que no tratan a las clases como «ciudadanos de primera»; otros, como Smalltalk u Objective C pueden recibir como parámetro el objeto que representa la clase a crear (en estos, las propias clases se comportan prácticamente como prototipos).

Consecuencias

● Inconvenientes:

- La implementación de la operación de clonación puede no ser fácil
 - ▶ Si las clases ya existían
 - ▶ Si incluyen otros objetos que no se pueden clonar
 - ▶ En presencia de referencias circulares

Implementación

- **Como ya hemos visto, es especialmente útil en lenguajes como Java o C++, donde las clases no son objetos**
 - (Aunque en ocasiones podemos emplear reflectividad)
- **Hay lenguajes que lo incorporan «de serie»:**
 - Self, JavaScript...

Implementación

● Cuestiones a tener en cuenta:

- Usar un gestor o registro de prototipos
 - ▶ Cuando los prototipos no son fijos sino que se pueden crear o eliminar dinámicamente
 - ▶ No es más que una especie de diccionario que devuelve el objeto prototípico apropiado para una clave dada
 - ▶ Los clientes pueden así modificar el sistema sin tocar el código

Implementación

● Cuestiones a tener en cuenta:

- Implementación de la operación de clonación
 - ▶ ¿Copia profunda («deep copy») o superficial («shallow copy»)?
- Inicialización de los prototipos

Otro ejemplo



Galería de documento de Word



Buscar plantillas locales y en línea

▼ PLANTILLAS

Todas

Mis plantillas

Vista Diseño del bloc de...

Vista Diseño de impresión

Calendarios

Currículos

Diseño de fondo

Etiquetas

Formularios coordinados

Propuestas

Varios

Vista Diseño de publica...

Boletines

Catálogos

Certificados

Folletos

Letreros

Menús

Pósteres

Programas

Prospectos

Tarjetas & Invitaciones

Documentos recientes

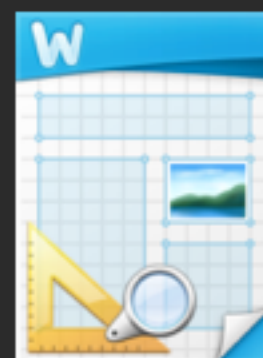
- Todas 10
- Hoy 0
- Ayer 1
- La semana pasada 2
- El mes pasado 3



Documento de Word



Diseño del B...tas de Word



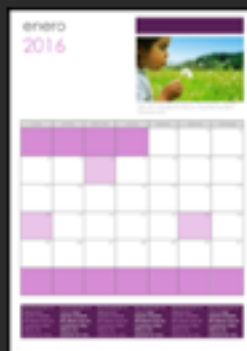
Diseño de p...nes de Word



Documento de Word



Calendario de diapositivas



Calendario de eventos



Calendario de titulares



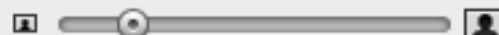
Calendario fotográfico



Calendario h...el primer día



Calendario h...el primer día



Cancelar

Elegir



Cancelar

Elegir

Otras cuestiones

Cuestiones

- ¿En qué se diferencia del *Factory Method*?
- ¿Y del *Abstract Factory*?