****

**CURSO DE TEORÍA DE LA COMPUTACIÓN Y LA COMPILACIÓN**

**2020-ll**

**TRABAJO FINAL Y MANUAL DE USUARIO PARA EL APLICATIVO**

Luis Felipe Moreno Chamorro

Kevin Danilo Arias Buitrago

Michael Stiwar Zapata Agudelo

Juan José Hurtado Álvarez

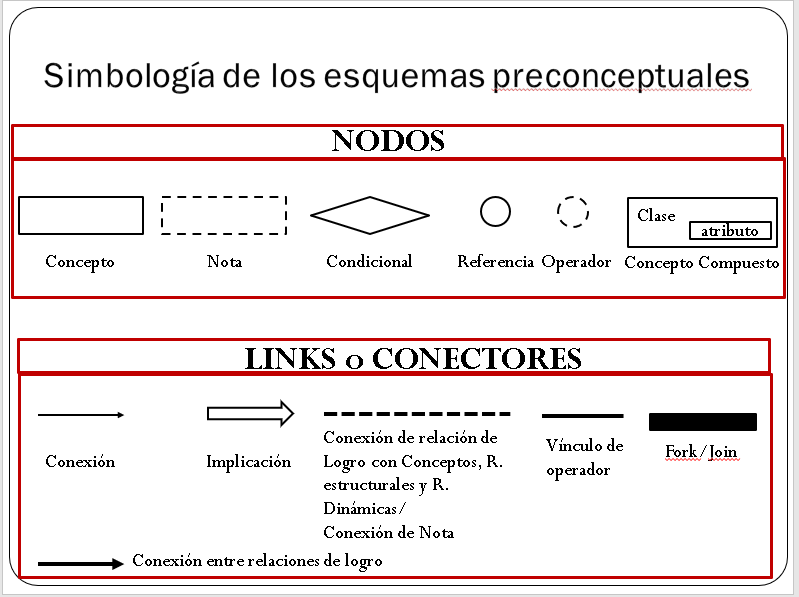
Julian Esteban Carvajal

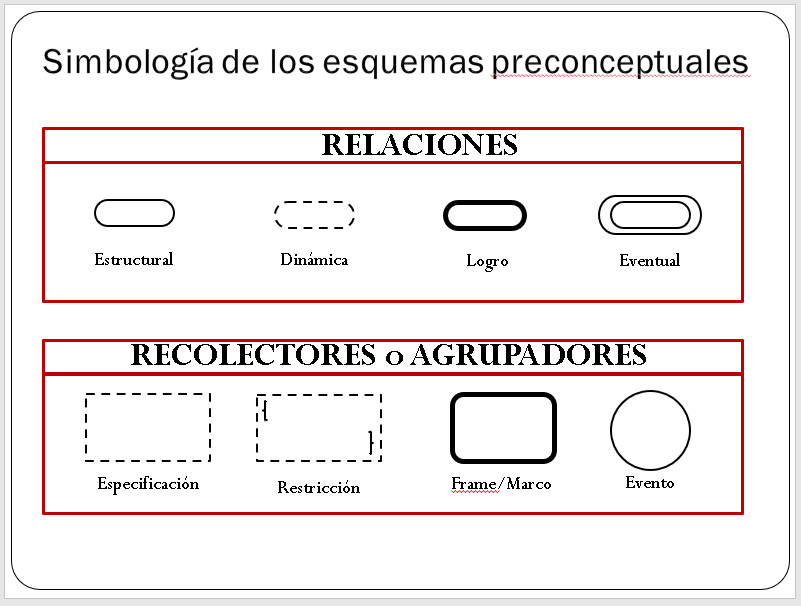
Federico Milotta

Juan Felipe Valencia

MODELADO:Esquema preconceptual

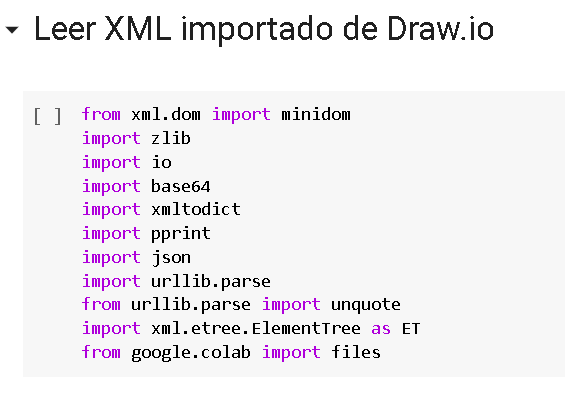
Usaremos como referencia la simbología de los esquemas preconceptuales, pero nos permitiremos modificaciones que permitan implementar de manera más adecuada el aplicativo. Las modificaciones serán oportunamente documentadas.

****

****

**LENGUAJE DE DESARROLLO:** Python

Esto se debe a que es un lenguaje versátil y flexible, cosa que se adecúa a nuestras necesidades. También por ser el lenguaje más común entre los integrantes del grupo, su fácil documentación e implementación. Se utilizará la herramienta Google Collaboratory para acceder fácil y ágilmente al código llevando seguimiento y guardando todo en el Drive de la Suite.



**LENGUAJE DE OBJETIVO:** Python Django

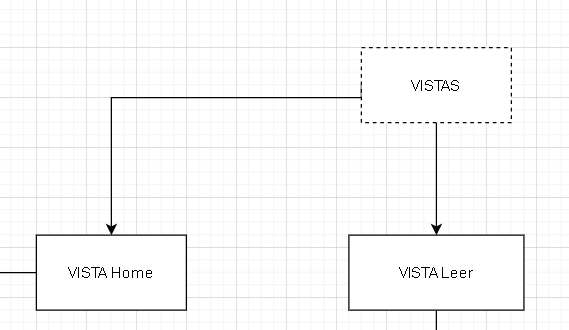
Para el lenguaje objetivo usaremos el framework Django porque varios integrantes tenemos conocimiento en él y además es una herramienta muy utilizada en la que podemos acceder a documentación muy fácilmente.

**REGLAS**

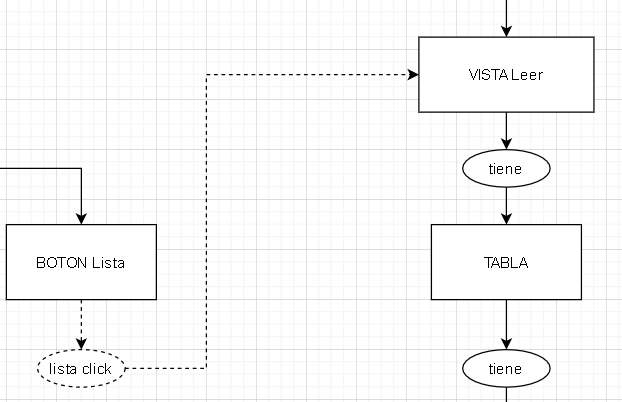
* En algunos conceptos similares se implementará algo parecido a la herencia de una clase abstracta, el formato que usaremos es el siguiente:

La clase abstracta irá en un cuadro punteado (como las notas) y de ahí saldrá una línea hacia los conceptos similares.

Esto no tiene implementación real en el código, sirve más como referencia para el modelo.

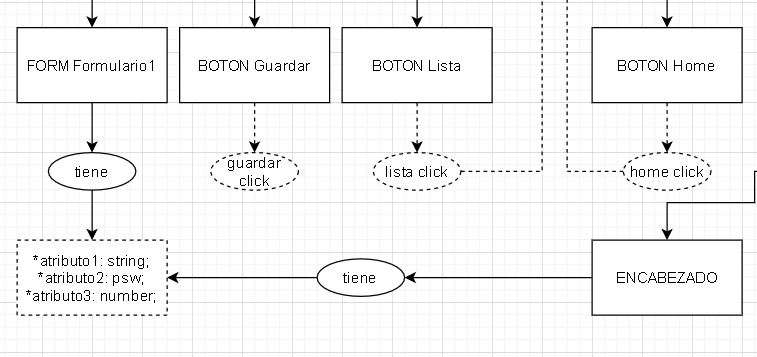


* Por motivos de simplicidad, cuando haya algún evento, este se denotará con un óvalo punteado y se conectará a los conceptos implicados en el evento por medio de líneas punteadas. Es importante resaltar que importa el orden de las flechas.

[](http://epreglas1)

* La forma de expresar que un concepto tiene atributos será un poco modificada, nos ahorraremos el concepto que especifique atributo, en cambio iremos directamente a la nota que los describe.

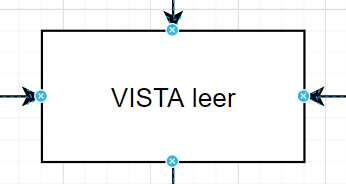
Estos campos de atributos pueden ser compartidos por conceptos



* **REGLA DE CONEXIÓN**:

En todos los casos que se conecten algún elemento con otro o con alguna flecha, se debe conectar correctamente, no se puede dejar sólo la conexión visible.

Para comprobar, cada elemento debe verse así al seleccionar las flechas que le entran o salen.

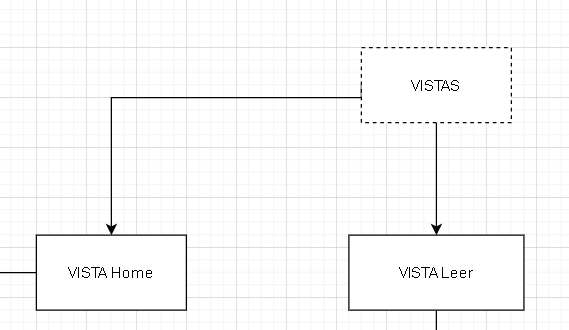


* **REGLA DE HERENCIA** (opcional, meramente visual):

En algunos conceptos similares se implementará algo parecido a la herencia de una clase abstracta, el formato que usaremos es el siguiente:

La clase abstracta irá en un cuadro punteado (como las notas) y de ahí saldrá una línea hacia los conceptos similares.

Esto no tiene implementación real en el código, sirve más para organizar estéticamente el modelo.

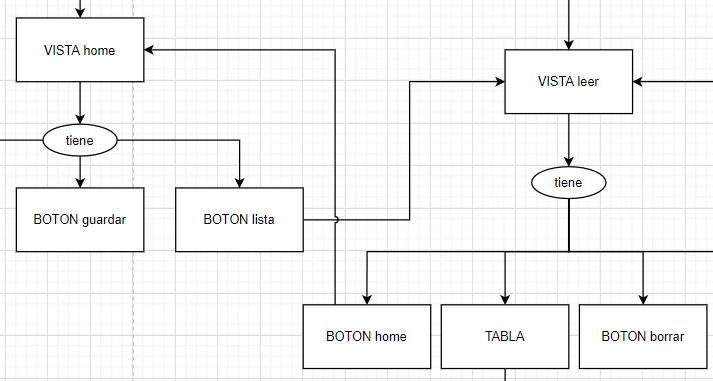


* **REGLA DE FLUJO:**

El diagrama completo respeta un flujo. Para tener un Formulario, una Tabla o un Botón se debe tener una Vista, cuando un Botón se conecta a una Vista, la flecha que los une debe de estar en la dirección Botón→ Vista

* **EVENTOS:**

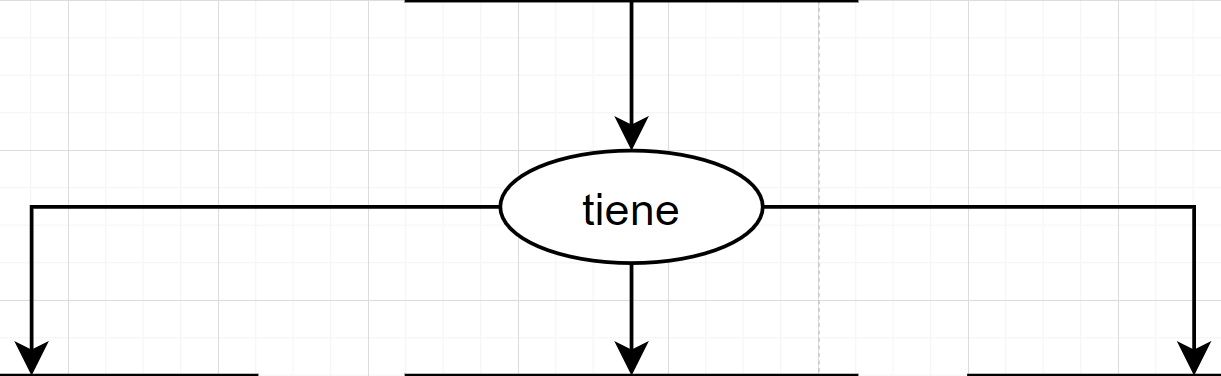
Los eventos se crean al conectar los elementos ‘BOTON’ a otro elemento. Para esto se debe respetar el flujo que tiene la flecha, es decir, la punta termina donde llega el evento. En el caso de que haya un botón que lleve a otra vista, se denotará con un cuadrado que comience con la palabra ‘BOTON’ seguido del texto que irá dentro de este, y con una flecha se denota la vista a la que lleva.



* **RELACIÓN ‘tiene’:**

Esta se denota como un óvalo de línea continua al que le entran flechas y le salen flechas, esta se usa como intersección para explicar la relación que tiene un elemento con otros.

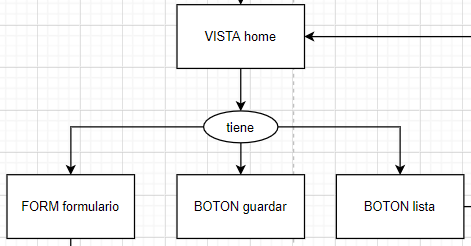
Esta relación al ser usada siempre debe tener un flujo que entra y n flujos que salen (n mayor a 0).



**REGLAS PARA CADA ELEMENTO:**

* **VISTAS:**

Las vistas se definen con la palabra ‘VISTA’ seguido del nombre de la vista dentro de un rectángulo de línea continua. Para denotar lo que tiene se conecta a un óvalo con la palabra ‘tiene’ y se conectan las flechas a los elementos.

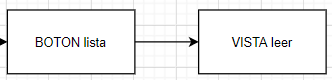


* **BOTONES:**

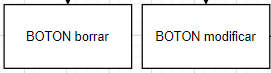
Los botones se denotan con la palabra ‘BOTON’ seguida del nombre dentro de un rectángulo de línea continua.

Hay dos tipos de botones dependiendo de su funcionalidad. Los que llevan a vistas y los que tienen funciones específicas.

* Los botones que llevan a otra vista se deben conectar usando una flecha que va desde el botón directamente a la vista.



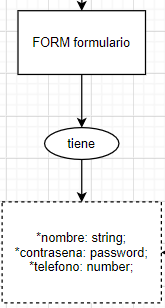
* Los botones que tienen funcionalidades específicas deben tener de nombre la funcionalidad que tienen, por ejemplo ‘guardar’, ‘borrar’, ‘modificar’, ‘editar’ o ‘cancelar’.



* **FORMULARIOS:**

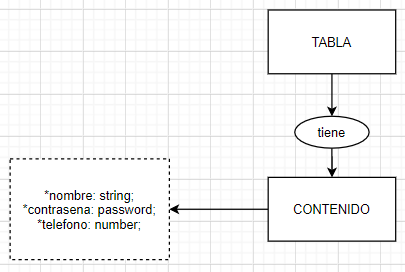
Los formularios se denotan con la palabra ‘FORM’ seguida del nombre del formulario dentro de un rectángulo de línea continua.

Al igual que en las vistas, estos elementos se deben conectar a una relación ‘tiene’ (como está expresado en la imagen, dentro de un óvalo de línea continua) que se conecta a un elemento ‘atributo’.



* **TABLAS:**

Las tablas se deben denotar sólo con la palabra ‘TABLA’ dentro de un rectángulo de línea contínua. Esta se conecta por medio de una relación ‘tiene’ al contenido, este después se conecta directamente a un elemento ‘atributo’ para demostrar las columnas que puede tener la tabla.

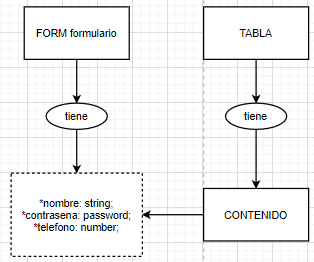


* **ATRIBUTOS:**

La forma de expresar que un concepto tiene atributos será un poco diferente al EP clásico, por ejemplo, se puede tener un formulario cuyos campos se denotan con un cuadrado de línea punteada (elemento ‘atributo’), cada campo debe respetar la siguiente sintaxis:

Primero se agrega un asterisco (\*), luego se escribe su nombre, dos puntos (:), el tipo de dato que puede ingresar y se debe terminar con un punto y coma (;).

Estos campos pueden ser compartidos por conceptos en diferentes vistas.



Un ejemplo de un CRUD funcional con una sola entidad que tiene nombre, contraseña y teléfono, se vería así. En este ejemplo se puede evidenciar que se tienen 3 vistas, la vista Home, la vista leer y la vista modificar.

