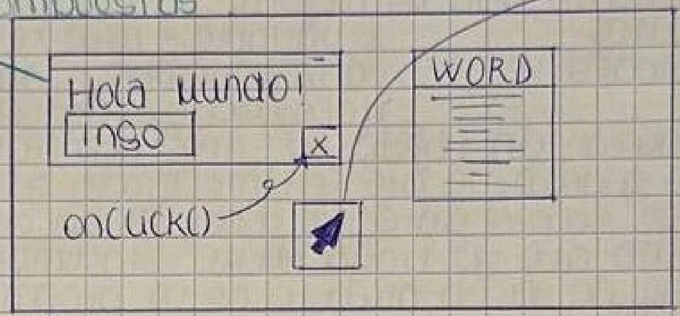
METODOLOGÍA DE LA PROGRAMACIÓN

GUI: INTERFACES GRÁFICAS DE USUARIO

En 1970, es el abandono del texto y su sustitución por interfaces gráficas más amigables. Algo que más o menos podamos usar todos sin conocer los detalles informáticos de la terminal. Esto es nuestro medio de comunicación con el ordenador.

System.out.println → Hola Mundo!

Tipos:
- Diálogos
- Fijos
- Flotantes
- Compuestas



VENTANA
- Visible
- Tamaño
- Posición

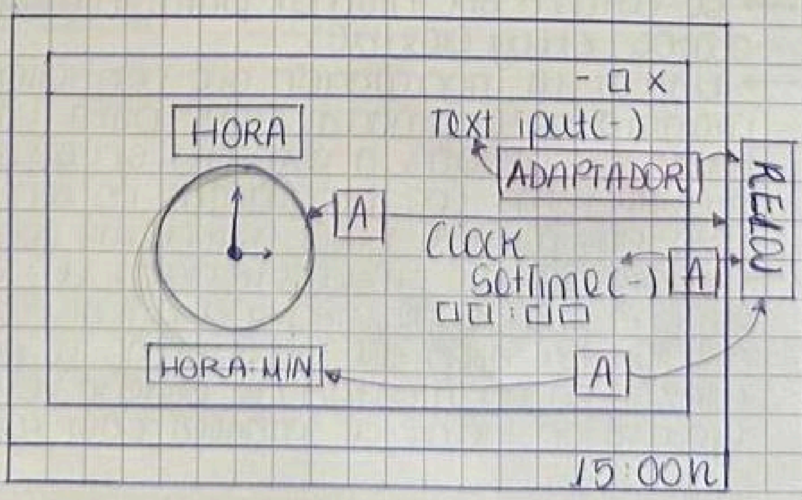
Una ventana dentro tiene una estructura:
- Titlebar → Botonera de estado
- Contenido
  - → Frame
    - → Text / Label
    - → Button

RATÓN
- Icono
- x, y
- Tamaño
- Pulsado 1
- Pulsado 2
- Pulsado 3
- Scroll
- Color
- dpi

El punto de todo esto es que tenemos un sistema muy complejo que requiere abstracción (sacar factor común): puntero del ratón, ventana con controles, es decir: POO.

Cuando un objeto llama a un método de otro objeto lo denominamos evento. Nuestras ventanas siempre están recibiendo todos los eventos, y para saber a cuáles ha de prestar atención se tiene el manejador de eventos (es algo así como nuestros oídos).

Esta manera de trabajar me lleva a un problema que es la anarquía. Antes teníamos el control de la secuencia de operaciones de manera que sabíamos qué había pasado antes y qué puede pasar después. Ahora no sé si el usuario ha metido el nombre, entonces necesito cubrirme de lo que pueda hacer o no el usuario. Para esto se han ido desarrollando estrategias: patrones de diseño software.




```
class Reloj {
  DateTime d
  String s
  TextInput t
  Clock c
  constructor
  void setD
      (DateTime h) {
    this.d = hora;
    t.setText(h);
    c.setTime(h);
  }
```

Patrón observador: cuando tienes una fuente de datos que puede cambiar y tienes que mostrarla en pantalla, es como un sistema de publicación y suscripción. Tú te asocias a un elemento que puede cambiar y (tienes que mostrarla en pantalla) si el elemento cambia tú te comprometes a avisarles.
Y luego están las clases TextInput y Clock que no programo yo para poder gestionarlos desde mi clase reloj, le añado un atributo suscribeIs y mi clase avisa a los suscriptores de mis cambios de estado y luego ya los suscriptores verán que hacen.

PERSISTENCIA: Cuando programamos en Java lo hacemos en base a unas clases que tienen una estructura.
una serie de atributos
una parte de métodos

```
X variable = new X()
variable.nombre = "Antonio"
Manejador.guardar(variable)
```

A la hora de inicializar un objeto lo cargo con datos, cuando lo tengo cargado lo guardo. Tendré un manejador.guardo(variable), pero lo que guardo es solo el dato, la información en un formato que sale del programa que yo haya hecho. Sin embargo el dato que yo guardo puede no ser un atributo de la propia clase sino un objeto de otra clase.
¿Como diferencio esto? La persistencia consiste en guardarlo y decidir cómo hacerlo. En muchos lenguajes los ficheros son estáticos, normalmente se utilizan registros y suelen tener un tipo determinado.
En la POO las clases (~~pueden~~) son grandes y pueden estar unas contenidas en otras, anidadas en vez de planas. Esto implica que los ficheros de datos en los que almacene mis objetos van a ser más complejos.

INTEROPERABILIDAD: Capacidad de objetos de diferentes lenguajes, plataformas o sistemas para comunicarse entre sí y trabajar juntos son formatos de ficheros que me van a permitir el intercambio de información sin rigidez, van a permitir adaptarse a múltiples problemas y van a permitir alterar la estructura propia del fichero de datos en el que estamos trabajando.
Los ficheros de datos pueden ser:
- Binarios: sistemas de registros (old-school)
- Texto: Txt (sin formato)
  CSV (valores separados por comas)
- Ficheros con interoperabilidad: estructura en la cual se puede extraer la definición de los datos que hay ahí dentro.
  * JSON → se utiliza en internet para intercambiar datos. (Muy flexible).
  * XML → Marcar la información que contienen de una manera determinada. Viene para las arquitecturas orientadas a servicio en estas arquitecturas los programas no son monolíticos. Esto nos permite hacer estructuras escalables.
  * YALM → contenedores (ing. estructura). Un contenedor es una especie de máquina virtual (conceptualmente). Es algo que desarrollas y puedes lanzar en un montón de ordenadores y se ejecuta en todos de manera casi igual.

Veamos ahora la estructura de estos ficheros de texto:

* JSON: Tiene su correspondiente en binario (JSONB) para que pueda tener un tamaño más pequeño y la mayor eficiencia, pues a pesar de que los ficheros de texto no tienen nos ofrecen la interoperabilidad.
  contiene un objeto, se asigna así:
  {Nombre: Antonio, Edad: 48}
  C: [{x: 0, y: 0, radio: 7}, {x: 1, y: 3, radio: 9}]
  Esto permite mandar este fichero a quien yo quiera, quien lee este fichero no necesita mi clase original.

* XML: Para definiciones especialmente precisas

```
<!-- XML ?/>
<x nombre = "Antonio", Edad = 48>
<C type = "Array" num = 2>
    <Circulo x = 0, y = 1, r = 7>
    <Circulo x = 1, y = 3, r = 9>
</C>
</x>
```

Otro ejemplo más especificado:

```
<!-- XML ?/>
<x>
  <nombre type = "String"> Antonio </nombre>
  <edad type: "int"> 48 </edad>
  <C type: "Array" num = 2>
    <Circulo>
      <x type: "int"> 0 </x>
      <y type: "int"> 1 </y>
      <radio type: "int"> 7 </radio>
    </Circulo>
  </C>
</x>
```

Para no estar repitiendo todo el rato los tipos de estructuras puedo usar DTD para definir primero los tipos. La estructura de definición nos facilita la búsqueda.
En grandes sistemas empresariales utilizan XML a nivel de programación se utiliza sobretodo JSON.

* YALM: con espacios de dos en dos

```
Tipo: x
  nombre: Antonio
  edad: 48
C:
  - x: 0
    y: 1
    radio: 7
  - x: 1
    y: 3
    radio: 9
```

Se utiliza para configuración, no meten datos y es muy limpio

Siempre puedes hacer en vez de un toString un toJSON y lo asignas igual ya que ya tienes un JSON (OPCIÓN A) API Reflection (OPCIÓN B): Le permite a java coger una clase y mirar cual es su estructura interna, de forma que ya nos lo escribe de la manera JSON. Uno de estos es el GSON. Además me permite coger un JSON y restaurar esa información en mi programa.

SERIALIZACIÓN: (es una especialización de la persistencia). Es el mismo proceso pero a nivel binario. Sirve para sistemas dentro del mismo lenguaje y permite la recreación completa de objetos. La idea no es la interoperabilidad sino el "intercambio entre texto y clase".