INTERFACES GRÁFICAS DE USUARIO (GUI)

En 1970, es el abandono del texto y su sustitución por interfaces gráficas más amigables (no necesitas saber la informática terminal donde se programa).

Intentamos hacer las cosas fáciles "dibujitos", "ratón",....

Este es nuestro medio de comunicación con el ordenador que ha ido variando y sigue.

Ejemplo: IHM, sistema de inteligencia artificial.

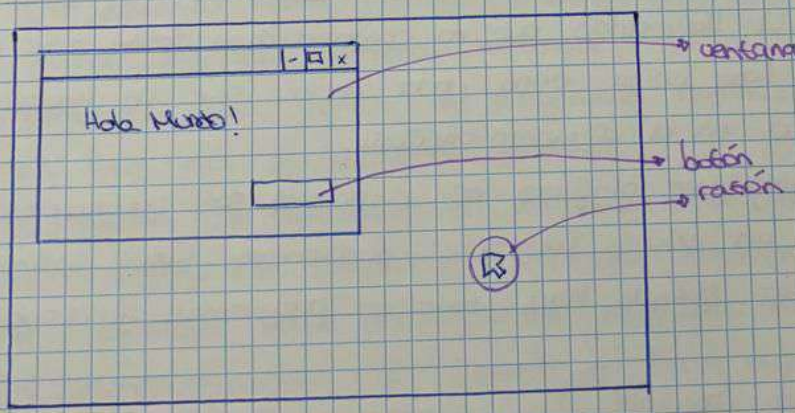
Queremos que en poco tiempo el ordenador nos hablee y no será necesario ni mirar (gafas, auriculares, ropa,...).

¿Qué tiene que ver con metodología de la programación? Nosotros hemos aprendido a hacer maneras de hacer programas. Hemos aprendido a hacer "cosas" y ahora tenemos que cosas. En informática, hemos tenido un uso del ordenador en base a texto, "programas". Ahora pasamos a una parte más gráfica, con una forma más abstracta orientada a objetos.

System.out.println("Hola Mundo!");

Antes tenía una pantalla donde salía texto, no nos preocupamos de qué pasaba por debajo porque era sencillo.

Ahora la cosa cambia mucho porque tengo una serie de abstracciones que antes no necesitaba tener.



Abstracciones:

1) Ratón: tiene un icono, color, una posición (x,y), sensibilidad (dpi), un tamaño, pulsado 1, pulsado 2, pulsado 3, scroll (rueda).

2) Ventana: es visible, tiene un tamaño y una posición. Tiene tipos:
   - Flotante
   - Fijas

- Compuestas
- Diálogos

Además, dentro de la ventana hay una estructura que tendrá un contenido

- Frame (Text/Label, button)

y una TitlePar (Barra de estado)

Tenemos que intentar eliminar los detalles mediante "sentido común". Tengo puntero de ratón, ventana, controles, ...

¿En programación cómo lo represento?

Si lo hago en Java, trato cada uno de los elementos como una clase y hago una instancia. Quiero un botón, pues de la clase botón hago botón información. Hacemos clases para representar los elementos que tengo en pantalla.

Gracias a la POO es relativamente sencillo poder organizar estas entradas.

Ejemplo: Ada no despegó, pero cuando aparecieron estos sistemas empezó a evaluarlos C y C++, evolución de C orientada a objetos. En este se descubrió la potencia de tener objetos abstrayéndote de los detalles y solo usando lo que tenemos que interesa.

¿Cómo funcionan los sistemas de interfaces gráficos de usuario?

En base a lo que se predefinió eventos. ¿Qué eventos hay? los que predefinimos en los objetos

Ejemplo: Puedo predefinir que mi botón tenga un evento que predefino onClick(). Hay cosas que no controlo como dónde está el ratón, pero sí hace cosa debo poder recibir que he hecho click y eso se recibe en el elemento onClick.

Estos sistemas (interfaces gráficos de usuario) son colaborativos porque no hay un único programa funcionando. Estos sistemas sólo tienen sentido cuando hay muchos programas funcionando a la vez.

Tú presentas una pantalla y esperas a que haga algo porque el usuario decide. Nosotros no tenemos control del ratón, sólo el usuario y es el puntero del ratón el que utilizo para comunicarse con tu ventana.

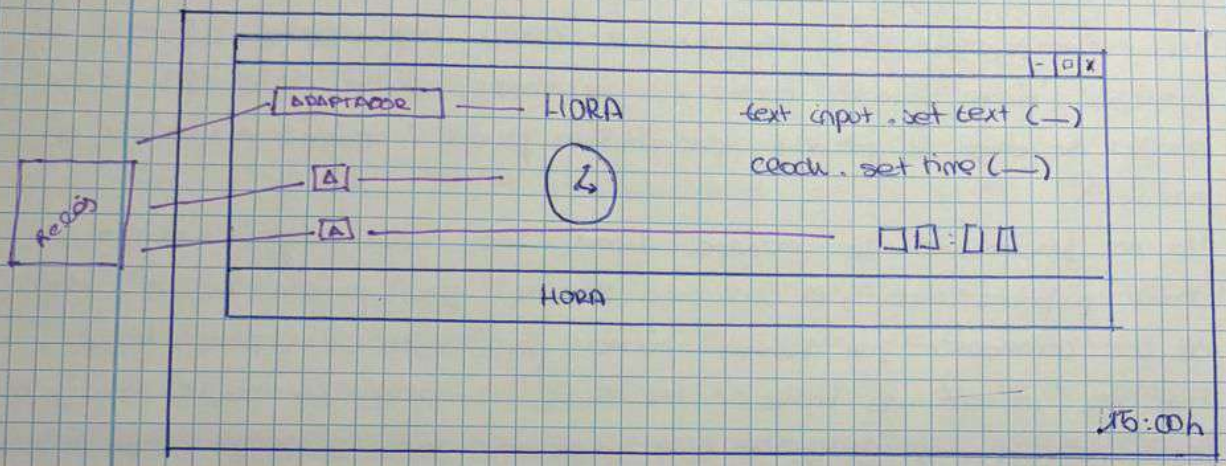
La ventana no está sola, puede tener más. Tu programa es una parte de otro programa mayor (en nuestro caso Windows).

Hasta ahora usamos las ventanitas de Windows para que me permita hacer y tener esos programas que queremos, no son nuestros, debemos adaptarnos a ellos.
Nuestro programa funciona y Windows también, (sistema operativo que usen), ese programa es el que tiene control de lo que se denomina entrada-salida, teclado-ratón, pantalla. Nuestro programa realiza peticiones de Windows y de ahí direcciones a Windows.
Cada vez que el usuario mueve el ratón mínimamente, Windows lo comunica a la ventana que hay por debajo y comunica donde está (la información) que llega a las distintas ventanas que llegan como eventos.
Todos los objetos tienen disponibles todos los eventos. Nuestras ventanas tienen siempre todos los eventos, pero la diferencia es a cuáles de ellos le prestes atención (se puede añadir manejadores de eventos.)
Ya no funcionamos de manera lineal sino donde hay un inicio donde preparas todo y el usuario decide qué puede hacer dentro de los controles que hemos hecho para que el usuario haga.
Toda esa manera de trabajar me lleva al problema de la anarquía. ¿Cómo organizo el flujo de los programas para que salga lo que debe ser?
Es más difícil porque:
1) Es mas difícil que antes
2) Tiene que cubrir lo que el usuario haga o deje de hacer
Con el tiempo hemos aprendido a tener estrategias a la hora de programar. (¿qué debemos hacer cuando...?) En programación se llaman patrones de diseño software/ patrones software.
Un patrón es una manera de programar dependiendo del problema que tengas. Hay tres familias de patrones.
- De creación: se ubicarían en las bases de creación de objetos.
- Estructurales: son parecidos a lo que denominamos estructura de datos. Te dicen como estructurar la arquitectura.
- De comportamiento: se dedican a hacer algo concreto.

ESTRUCTURA:
- Qué quiero hacer
- Problema
- Como aplicarlo
- Solución

Algunos ejemplos:

- Cadena de responsabilidad: das 1 problema y no necesitas saber cuál lo resuelve, sino que pasa de 1 a otro.
- Patrón comando: una funcionalidad se encapsula dentro de una clase con lo mínimo para ejecutarse. Sirve para instanciarla y meterla donde tú quieras
- El observador: sirve cuando tienes una fuente de datos que puede cambiar y tienes que mostrarla en pantalla. Te asocias a un elemento que puede cambiar y cuando cambie él se comprueba a quien de que ha cambiado.




```
class Reloj {
    Date time d
    String s
    TextInput T
    Clock C
    // constructor
    Void setD (DateTime hora) {
        this.d = hora
        this.setText (hora)
        C.setTime (hora)
    }

Main {
    T
    C
    Reloj (T,C)
}
```

Sistema de adaptadores, patron MVC modelo vista controlador.
¿Qué hace cada uno de ellos?
El modelo son los datos == Mi relais.
La vista es lo que representa los datos == dibujitos.
El controlador se encarga de realizar las actualizaciones != adaptador.
Funcionamiento:
Decide hacer una actualización de datos, cuando el modelo cambia, notifica a la vista y es esta la que accede a los datos.
Este modelo es así porque a su vez la vista está conectada con el controlador para que la vista pueda operar. Debe decir a otro ejecutar las acciones, es decir, el controlador. Cuando quiere que pase algo, le hace al controlador no la vista. Este patron se utiliza en la web.
View == página web.
Hacer esto, le ofrece la información al controlador que verifica que todo está bien, cambia la base de datos (en model) de lo que está haciendo la vista dice que ha cambiado y él le pide como al modelo y lo cambia.
Se usan para intentar reducir los problemas que se pueden tener y si no lo usas es un infierno porque es más lioso que con el esquema MVC.

PERSISTENCIA.

En Java lo hacemos en base a unas clases.

```
Class Circle {
    Int x
    Int y
    Int radio
}
Class x {
    // Atributos
    String nombre
    Int Edad
    Circle [] c
    // Métodos
    Void dimeNombre () {

    }
{
X variable = new X();
Variable.Nombre = "Antonio"
```

Cada vez que tengo un objeto inicializado, lo guardamos para no tener que volver a hacerlo.

Manejador.guardar (Variable) // Guarda la variable de la clase X que yo he creado. (Guarda Antonio, el dato).

¿Es tipo de dato que yo guardo no tiene porque ser un atributo de una clase sino es objeto de una. La persistencia consiste en guardarlo y decidir donde hacerlo. En muchos lenguajes los ficheros son fijos. Si decimos que un fichero es de clase X, solo se pueden guardar cosas de este tipo.

Antiguamente, especificabamos antes de que tipo queríamos el fichero (la estructura) y solo podíamos escribir en este fichero cosas de esa misma estructura. Si lo abrimos y decimos que tiene estructura diferente, entonces lo leería mal y nos daría datos erróneos o no permitirnos que funcione.

Hoy en día, guardamos objetos y no pensamos en ellos como estructuras de datos que van a disco sino conjuntos de datos que existen dentro de mi programa en distintas fases (almacenado, en red, ...).

En programación orientada a objetos, las clases son más grandes (en el pasado nos centrábamos en el espacio y había estructuras pero ya que era muy importante el espacio) esto indica que ya no puede ser tan eficiente buscando en fichero.
Hoy en día, tenemos una clase con diferentes arrays, listas, claves, etc. ¿Cómo nos apañamos para meterlo en un fichero de datos?
Los ficheros serán más complejos (en mundo informático se ha hecho más grande y por tanto menos "controlado"). Dentro de los formatos nuevos hay formatos de interoperabilidad donde se busca definir conmatos flexibles.

INTEROPERABILIDAD.

Los conmatos son formatos de ficheros que permiten el intercambio de información entre comillas sin rigidez. Son formatos de fichero que permiten adaptarse a múltiples problemas y permiten alterar la estructura propia del fichero de datos que estamos trabajando.
Primero hay que pensar donde nos estamos moviendo.
Un fichero de datos puede ser binario o de texto. En los binarios tenemos los sistemas de registros (lo de antiguamente, tengo una variable y la meto en el fichero.). En los de texto hemos tenido muchos tipos

1) Txt (ficheros abiertos donde coges un editor de texto y ves lo que meses) = test vacío.

2) CSV (líneas separadas por comas) = pones en filas el valor de una variable coma otro valor coma ..., no se autodefinen, no dicen que estructura tienen internamente, no sabes porque está ahí.

3) EDI = ficheros abiertos con editor dentro y lees lo que hay dentro.

4) Interoperabilidad = estructura donde se puede saber los datos que hay ahí dentro.

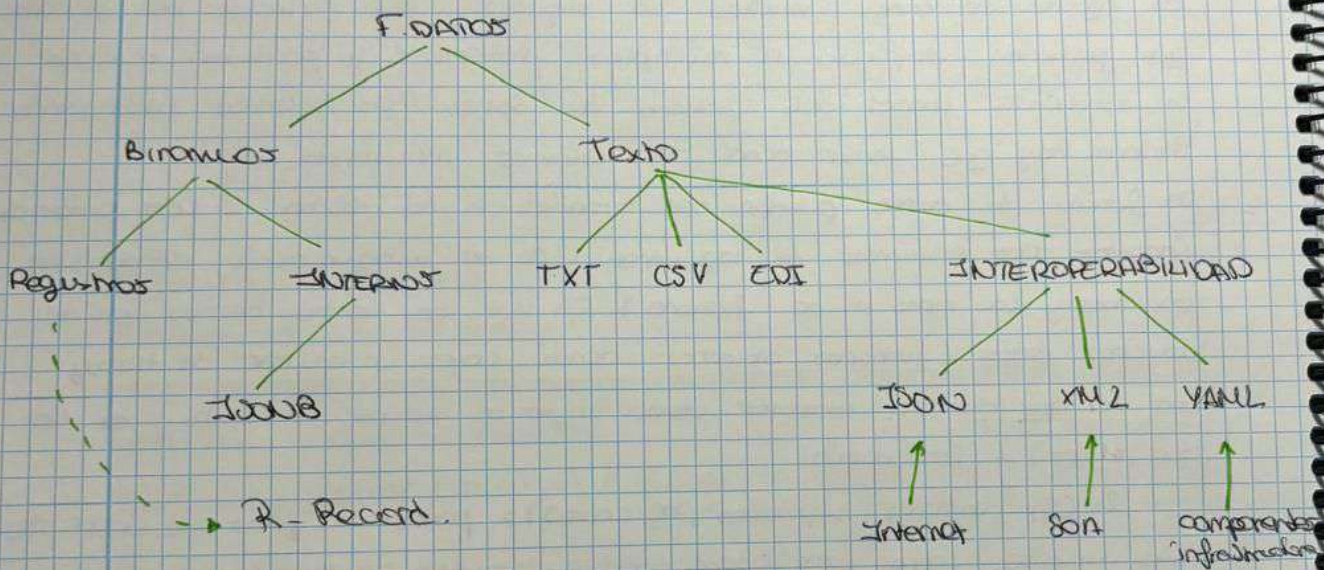
Todos son de texto, algunos tienen su correspondencia en binario

→ JSON (Notación de objeto de Java Script) = se utiliza sobre todo en internet (intercambiar datos). También sirve para todo tipo de ficheros especialmente de configuración no solo de internet.
Tiene su correspondiente en binario que se llama JSONB (para buscar la eficiencia)

→ XML (lenguaje de marcado extensible) = viene de los documentos de marcado y para las arquitecturas orientadas a servicio (SOA). En esta arquitectura hacemos funcionalidades que se hacen por separado

y entre ellas se conectan y entre ellas se conectan y hablan y son escalares, nos viene bien para internet ya que la manera de crear los cosas tecnológicamente es muy asincrona. Pensado para miles de usuarios que puedan crecer. Los programas intentamos automatizar en servicios y pasos de unos a otros por XML.

→ YAML (yet another markup language) = Trabaja con contenedores e infraestructura. Un contenedor es una especie de máquina virtual conceptualmente. Es algo que desarrollas y puedes tenerlo en montón de ordenadores y se ejecuta en todos más o menos igual pero hay que configurarlo. Para que sea más o menos limpio, usamos YAML.



JSON:

Un fichero JSON sere 1 objeto.

Se define:

JSON

{nombre: Antonio;
Edad: 48;
C: [{x=0, y=1, radio=7},
{x=1, y=3, radio=9}]
}

Esto me permite mandar este fichero a quien yo quiera, quien use este fichero no necesita mi clase original.

XML:

Pensado

<!-

<x

</x

otro

<!-

<x

XML:

Pensado para hacer definiciones que puedan ser extremadamente precisas.

<!-- XML? />
<x nombre = "Antonio" Edad: 48>
    <c type = 'Array" num =2>
        <circulo x=0, y=1, radio=7>
        <circulo x=1, y=3, radio=9>
    </c>
</x>

Otro ejemplo de JSON mas 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 dos tipos de estructura puedo usar CTC para definir primero dos tipos. La estructura de definición nos facilita la búsqueda.

En grandes sistemas empresariales se utilizan XML a nivel de programación se usa sobre todo JSON.

YAML:

Tipo: X
-- nombre: Antonio
-- edad: 48
-- C:
---- -X: 0
     Y: 1
     Radio: 7
     -X = 1
     Y = 3

Reto : 9

No se suele meter datos.

Opción 1:

Hacer una clase JSON y meter en otra clase hijo para usarlos.

Opción 2:

Java es un lenguaje orientado a objetos con mucha información por debajo. Tiene la API REFLECTION que le permite a Java coger una clase y ver su estructura interna.

Coge una clase y la transforma GSON (Google) y es capaz de hacer el camino contrario. Le das un JSON le dices la clase que quiero que restaure y lo hace.

SERIALIZACIÓN:

Es el mismo proceso, pero a nivel binario, se hace dentro del mismo lenguaje y permite la recreación completa de objetos. Dentro de un mismo sistema, dentro de un lenguaje completo. La idea es un intercambio entre texto y clase.

Marshalling, hacen una transformación de datos de un formato a otro (proceso complejo).