



Programación

07 Programación Interactiva e Interfaces Gráficas

José Luis González Sánchez



Contenidos

1. Patrón Observer
2. Eventos
3. Interfaz de Usuario
4. Prototipos
5. Guías de diseño
6. Programación interactiva

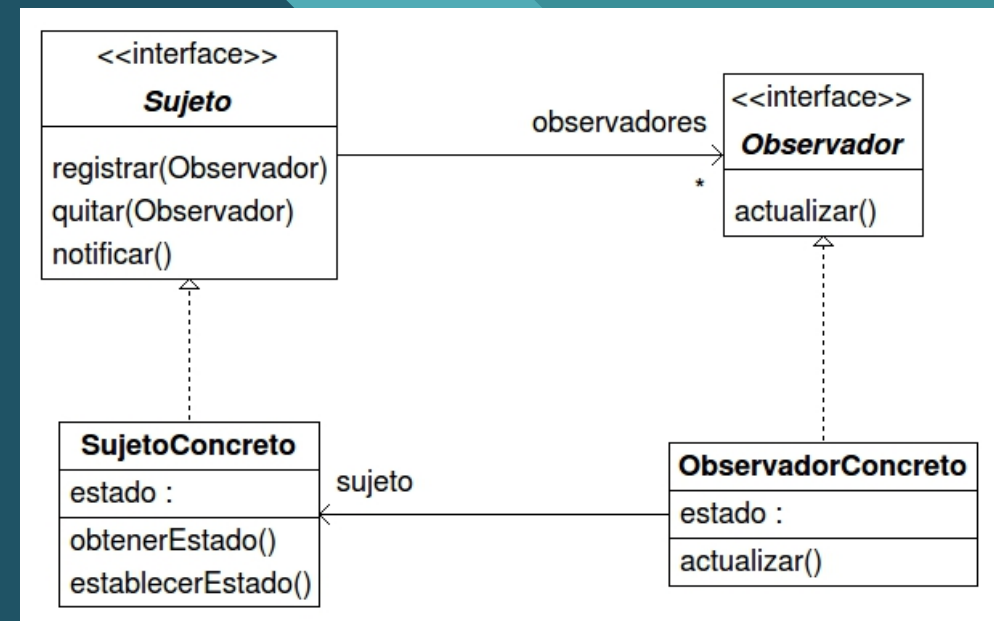


Patrón Observer

El principio de todo

Patrón Observer

- El patrón de diseño Observer, Observer Pattern o patrón observador es uno de los patrones de diseño de software más populares. Esta herramienta ofrece la posibilidad de definir una dependencia uno a uno entre dos o más objetos para transmitir todos los cambios de un objeto concreto de la forma más sencilla y rápida posible. Para conseguirlo, puede registrarse en un objeto (observado) cualquier otro objeto, que funcionará como observador. El primer objeto, también llamado sujeto, informa a los observadores registrados cada vez que es modificado.
- Algunos de los casos de aplicación típicos de este patrón son las GUI (interfaces gráficas de usuario), que actúan como interfaz de comunicación de manejo sencillo entre los usuarios y el programa. Cada vez que se modifican los datos, estos deben actualizarse en todos los componentes de la GUI. Esta situación es perfecta para la aplicación de la estructura sujeto-observador del patrón Observer. Incluso los programas que trabajan con conjuntos de datos en formato visual (ya sean tablas clásicas o diagramas gráficos) pueden beneficiarse de la estructura de este patrón de diseño.



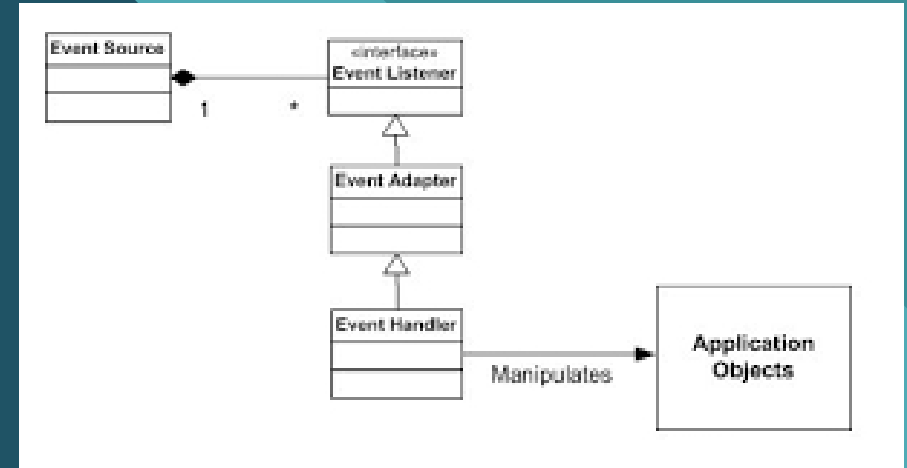


Eventos

Respondiendo a acciones que pasan

Eventos

- La programación dirigida por eventos, es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.
- Los eventos son acciones o sucesos que se generan en aplicaciones gráficas definidas en los componentes y ocasionado por los usuarios, como presionar un botón , ingresar un texto, cambiar de color, etc.
- Los eventos le corresponden a las interacciones del usuario con los componentes. Es decir, estamos observando la interacción del usuario
- Los componentes están asociados a distintos tipos de eventos, es decir, son observados desde distintos puntos de vista a nivel interactivo: si se pulsa una tecla, si se hace clic...
- Un evento será un objeto que representa un mensaje asíncrono que tiene otro objeto como destinatario, listener

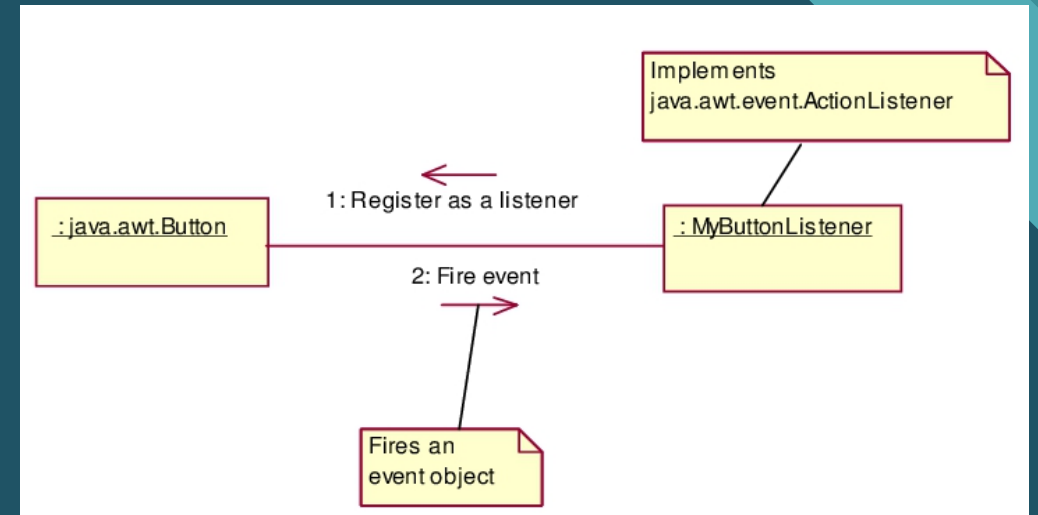


Eventos

- Por lo tanto, un objeto permite tener distintos Listeners que se encargan de observar distintos cambios a nivel interactivo: pulsar un botón, hacer clic... Se los debemos añadir, es decir, suscribirnos a ellos
- Los Listeners se encargan de controlar los eventos, esperan a que el evento se produzca y realiza una serie de acciones. Según el evento, necesitaremos un Listener que lo controle.
- Apartir de aquí el propio manejador del evento una vez avisado ejecuta la acción indicada.

```
componente.add"tipo evento"(new "tipo evento"){  
    metodos del evento ( tipo de evento) {  
        Acciones a ejecutarse  
    }  
});
```

```
boton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        texto.setText("¡DAM!"); }  
});
```





Interfaz de Usuario

Interactuando con nuestra app

Interfaz de Usuario

- La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface), utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles. Su principal uso consiste en permitir la comunicación con entre el usuario y la aplicación.
- Pueden ser:
 - Gráfica
 - Háptica
 - Por voz
 - Táctil
- Es importante optimizar el dialogo con el usuario/a y usar las metafora correctas.





Prototipos

Antes de programar nuestra interfaz, diseñala

Prototipos

- Un modelo o prototipo de UI es una simulación interactiva y real de cómo funcionará una aplicación, ya que permite ver los flujos de entradas y salidas, ya sea de una sola sección o de un producto digital completo.
- Los prototipos de UI nos ayudan a ordenar las ideas, explorar diferentes caminos de concepto o diseño, y detectar posibles problemas o carencias antes de empezar la fase de programación de nuestro proyecto digital. También nos permiten saber cómo diseñar los menús o la navegación entre las funcionalidades de nuestra aplicación.
- Con esto, puedes probar, analizar, ajustar y perfeccionar los elementos con los que van a interactuar los usuarios de tu aplicación.
- Tipos
 - Alta o baja fidelidad
 - Horizontal o vertical

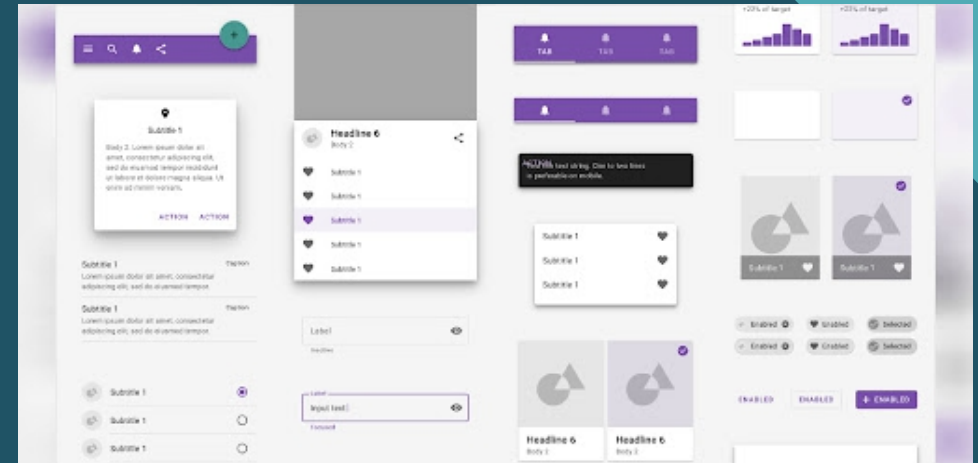
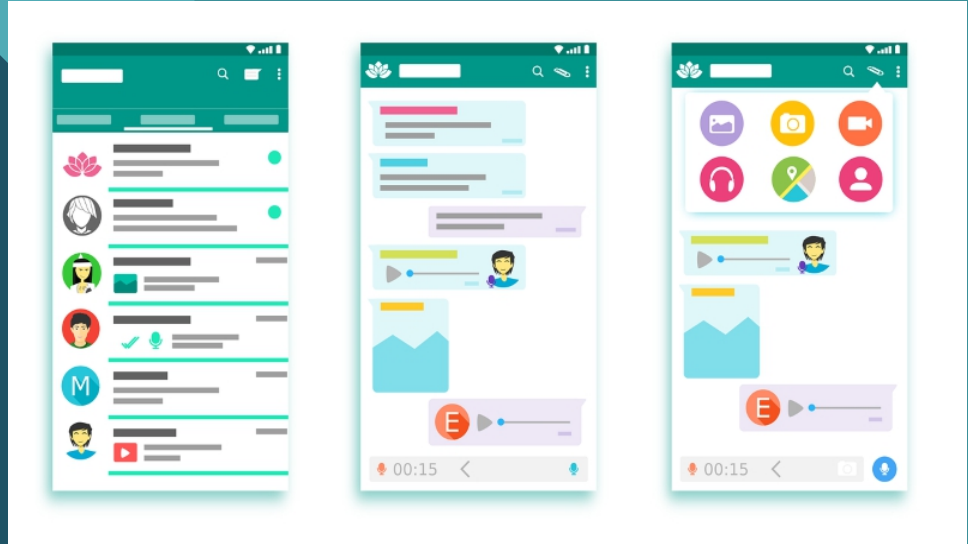
The screenshot shows a web application titled 'PublicPS'. At the top, there is a navigation bar with links: 'Perfil', 'Procesos', 'Grupos', and 'Salir'. Below this is a sub-navigation bar with buttons: 'Nuevo' (highlighted), 'Grabar', 'Editar', and 'Cerrar'. A tabbed interface follows with tabs: 'Datos Proceso', 'Informativo', 'Participativo', 'Evaluación', and 'Publicar'. The 'Datos Proceso' tab is active, displaying a form with the following fields: 'Título:' (text input), 'Descripción:' (text input), 'Contenido:' (large text area), and 'Medio de publicación:' (a dropdown menu currently showing 'Web'). On the left side of the form, there is a sidebar titled 'Mis procesos' containing three expandable sections: 'Activos' (with 'Proceso1' and 'Proceso2'), 'Publicados' (with 'Proceso2'), and 'Finalizados' (with 'Proceso3').

Guías de Diseño

Haz y muestra lo esperado según el sistema donde estés

Guías de diseño

- Las guías de diseño son guías o un compendio de recursos que los fabricantes de los sistemas ofrecen con el objetivo de crear mejores experiencias interactivas al ofrecer patrones de diseño, recursos, guías de colores, fuentes, etc.
- De la misma manera las guías nos sirven como mecanismo para verificar y validar nuestros diseños, pues seguirlas nos asegura que el usuario/a comprenda las metáforas que mostramos en nuestra interfaz y con ello aumente la efectividad, eficiencia, aprendizaje y satisfacción a la hora de usar nuestro producto software.
- Antes de diseñar cualquier interfaz, analiza la guía de estilos o diseño y usa elementos, colores, fuentes y disposiciones recomendadas.
- Ejemplos:
 - Material
 - Windows
 - Apple

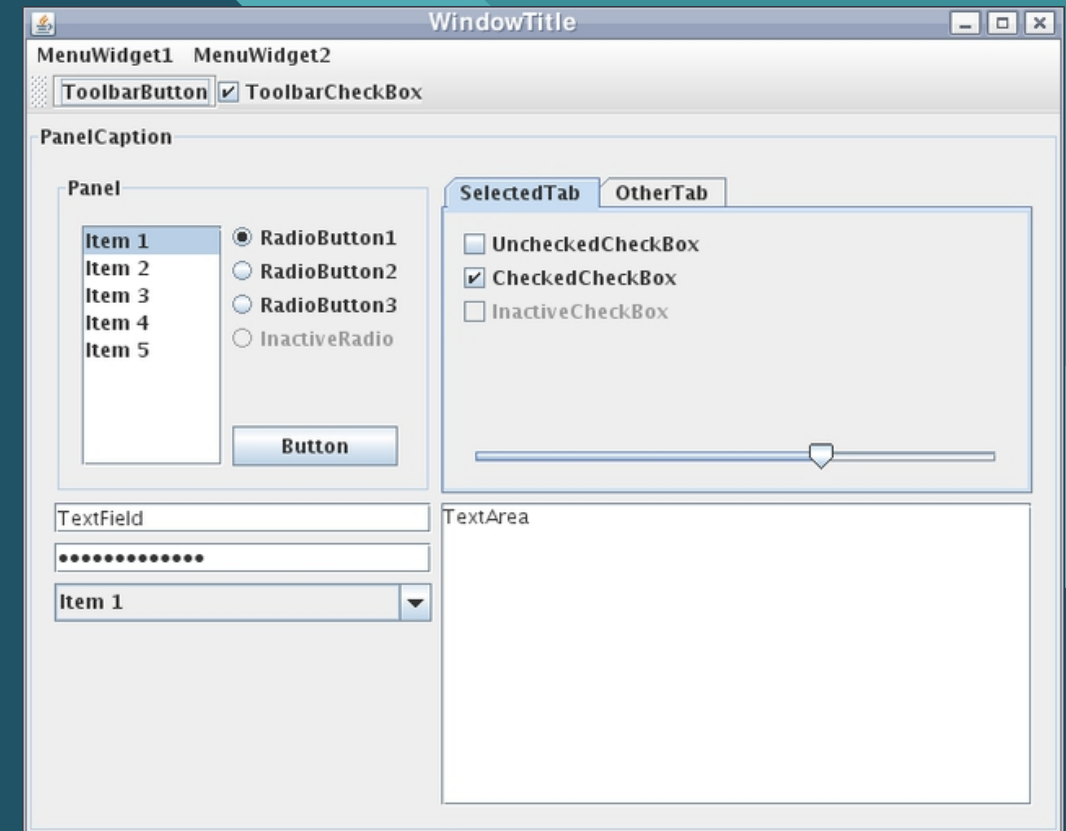


Programación Interactiva

Constriyendo nuestra apliaciones interactivas

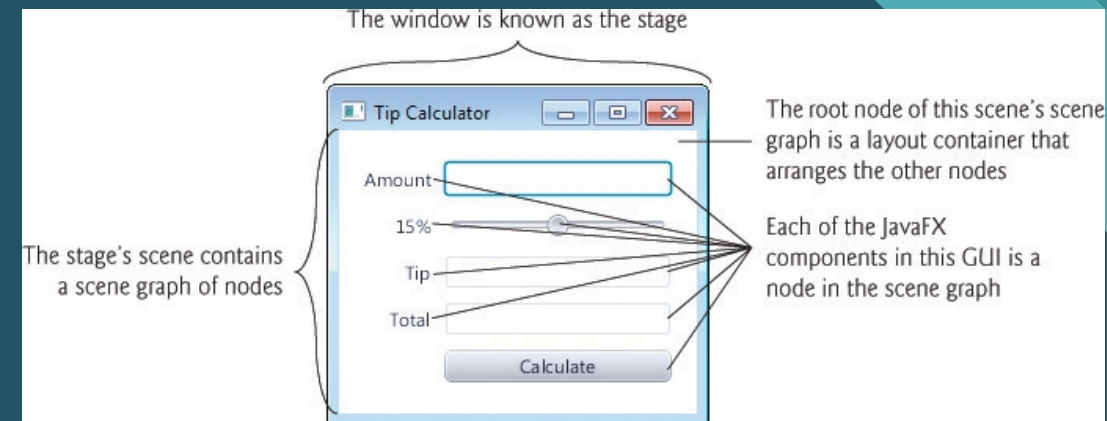
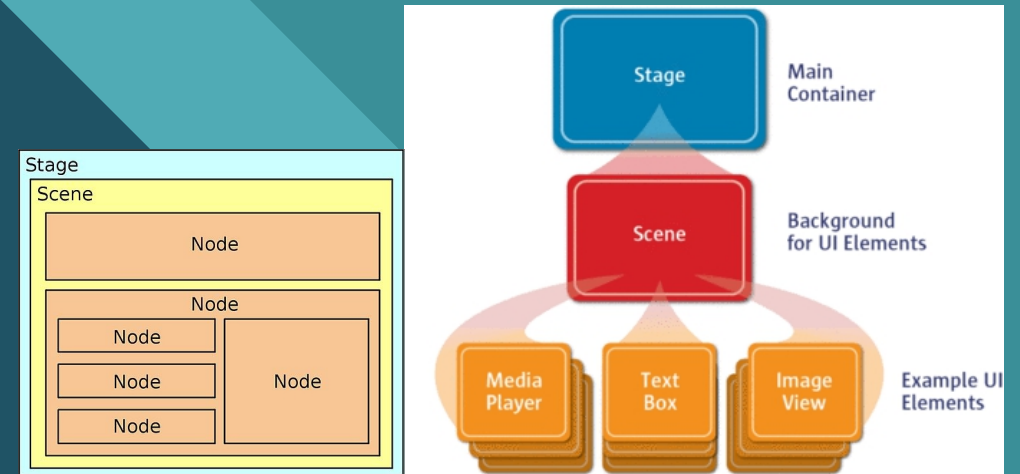
Java Swing

- Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas.
- Es un framework MVC para desarrollar interfaces gráficas para Java con independencia de la plataforma. Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:
- Independencia de plataforma.
- Extensibilidad: es una arquitectura altamente particionada: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Personalizable: dado el modelo de representación programático del framework de Swing, el control permite representar diferentes estilos de apariencia "look and feel" (desde apariencia MacOS hasta apariencia Windows u otra propia).
- Esta siendo sustituido por otras alternativas.



Java FX

- JavaFX nos ofrece la creación de Rich Internet Applications (RIAs), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas.
- Las aplicaciones JavaFX pueden ser ejecutadas en una amplia variedad de dispositivos, además puede integrarse código Java en programas JavaFX. JavaFX es compilado a código Java.
- Fundamentos:
 - Stage: Es como se llamará al lugar donde se visualizará nuestra pantalla, podría ser como el escenario de una obra de teatro. Puede ser la ventana.
 - Scene: Es la escena o vista en ese momento que hay en la pantalla, podemos cambiar entre ellas dentro del mismo escenario. La Escena contiene un grafo de nodos de manera jerarquizada.
 - Root Node: es el nodo raíz o contenedor principal donde se dispondrán el resto de nodos o elementos interactivos.
 - Nodos: son el conjunto de elementos interactivos y/o gráficos que aparecen en una escena.



Java FX

FXML es un lenguaje de marcas que nos permite definir las interfaces de usuario y con ello crear nuestras vistas.

A partir de estos ficheros, podemos realizar el binding o enlazamiento entre los elementos gráficos y los objetos del código.

Es importante que diseñemos estos ficheros siguiendo las guías de diseño de los fabricantes o sistemas a donde vamos destinados. Para ello podemos hacer uso de CSS a nivel global o para cada nodo de la escena.

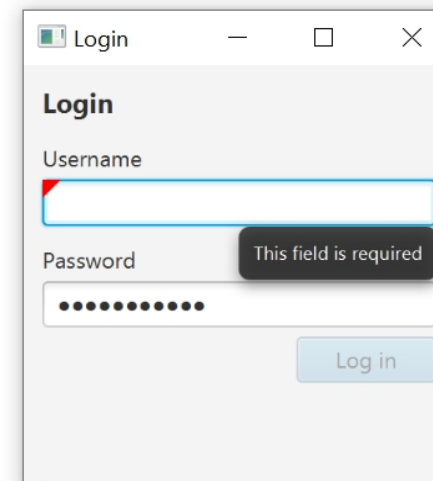
Podemos usar Scene Builder o el propio Scene Builder de IntelliJ.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import java.lang.*?>
4  <?import java.util.*?>
5  <?import javafx.scene.*?>
6  <?import javafx.scene.control.*?>
7  <?import javafx.scene.layout.*?>
8
9  <AnchorPane id="AnchorPane" prefHeight="200" prefWidth="320" :
10  <children>
11      <Button layoutX="126" layoutY="90" text="Click Me!" on
12      <Label layoutX="126" layoutY="120" minHeight="16" minW
13  </children>
14  </AnchorPane>
15
```

Tornado FX

Tornado fX es un framework para kotlin que te permite usar Java fX de manera declarativa. Que es definir una interfaz declarativa, pues es tan simple, como que tú dices lo que quieres y no te centras tanto en cómo lo quieres. Luego aplicandohojas de estilo o estilos de plataforma obtienes la interfaz final.

```
class LoginForm : View( title: "Login") {  
    val user = UserModel()  
  
    override val root = form { this: Form  
        fieldset(title, labelPosition = VERTICAL) { this: Fieldset  
            field( text: "Username") { this: Field  
                textfield(user.username).required()  
            }  
            field( text: "Password") { this: Field  
                passwordfield(user.password).required()  
            }  
            buttonbar { this: ButtonBar  
                button( text: "Log in") { this: Button  
                    isDefaultButton = true  
                    enableWhen(user.valid)  
                    action { user.login() }  
                }  
            }  
        }  
    }  
}
```

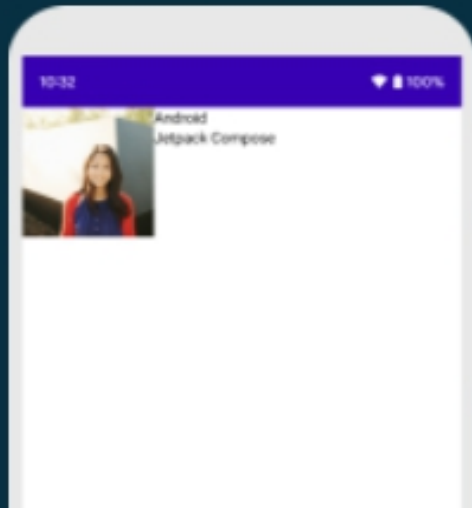


Compose

Librería inicialmente creada por Google para crear interfaces reactivas y declarativas en Android. Posteriormente el proyecto es tomado por JetBrains para llevar la filosofía Compose a escritorio y a web. De esta manera se busca programar tu interfaz y aplicación una vez y que se ejecute y se adapte su interfaz al dispositivo de destino.

```
@Composable
fun MessageCard(msg: Message) {
    Row {
        Image(
            painter = painterResource(R.drawable.profile_picture)
            contentDescription = "Contact profile picture",
        )

        Column {
            Text(text = msg.author)
            Text(text = msg.body)
        }
    }
}
```

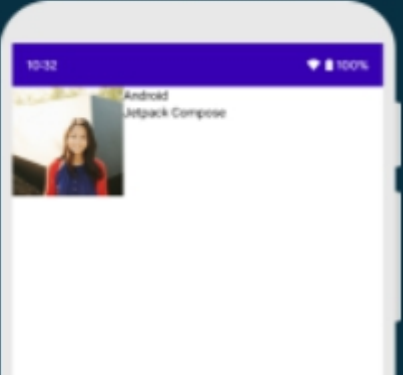


Compose

Librería inicialmente creada por Google para crear interfaces reactivas y declarativas en Android. Posteriormente el proyecto es tomado por JetBrains para llevar la filosofía Compose a escritorio y a web. De esta manera se busca programar tu interfaz y aplicación una vez y que se ejecute y se adapte su interfaz al dispositivo de destino.

```
@Composable
fun MessageCard(msg: Message) {
    Row {
        Image(
            painter = painterResource(R.drawable.profile_picture)
            contentDescription = "Contact profile picture",
        )

        Column {
            Text(text = msg.author)
            Text(text = msg.body)
        }
    }
}
```



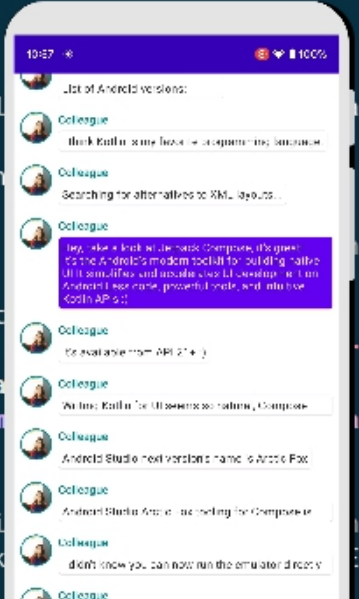
<https://developer.android.com/jetpack/compose?hl=es-419>

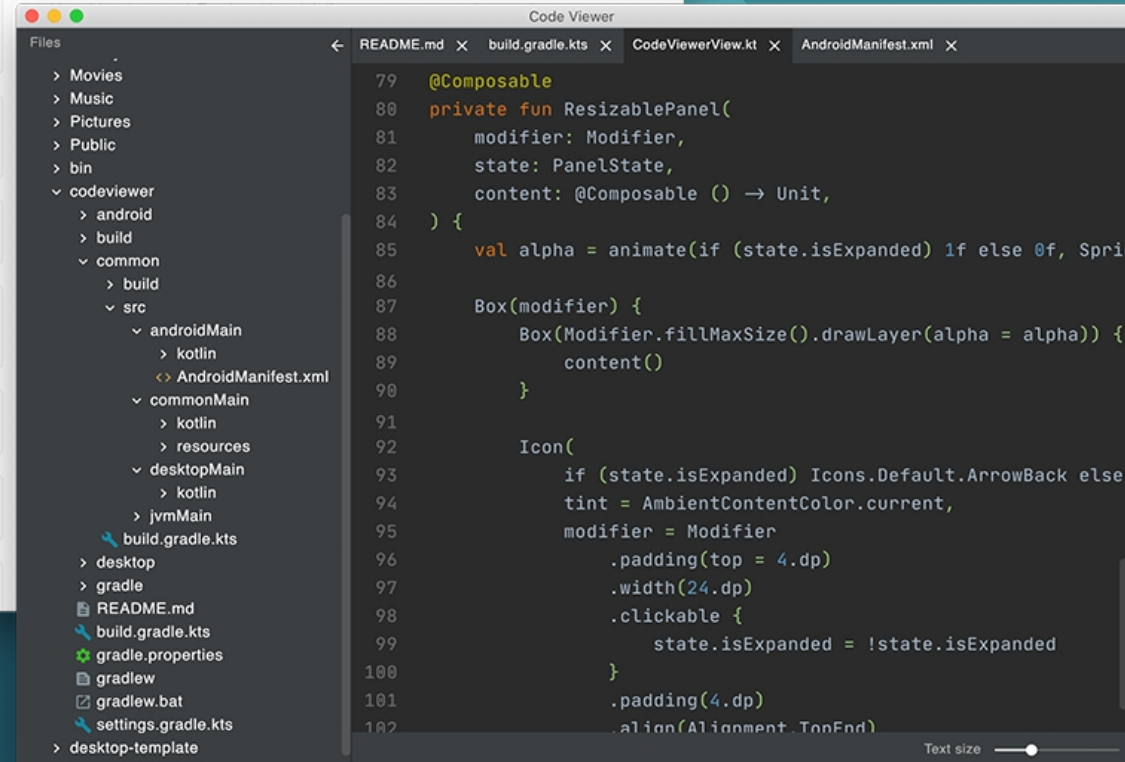
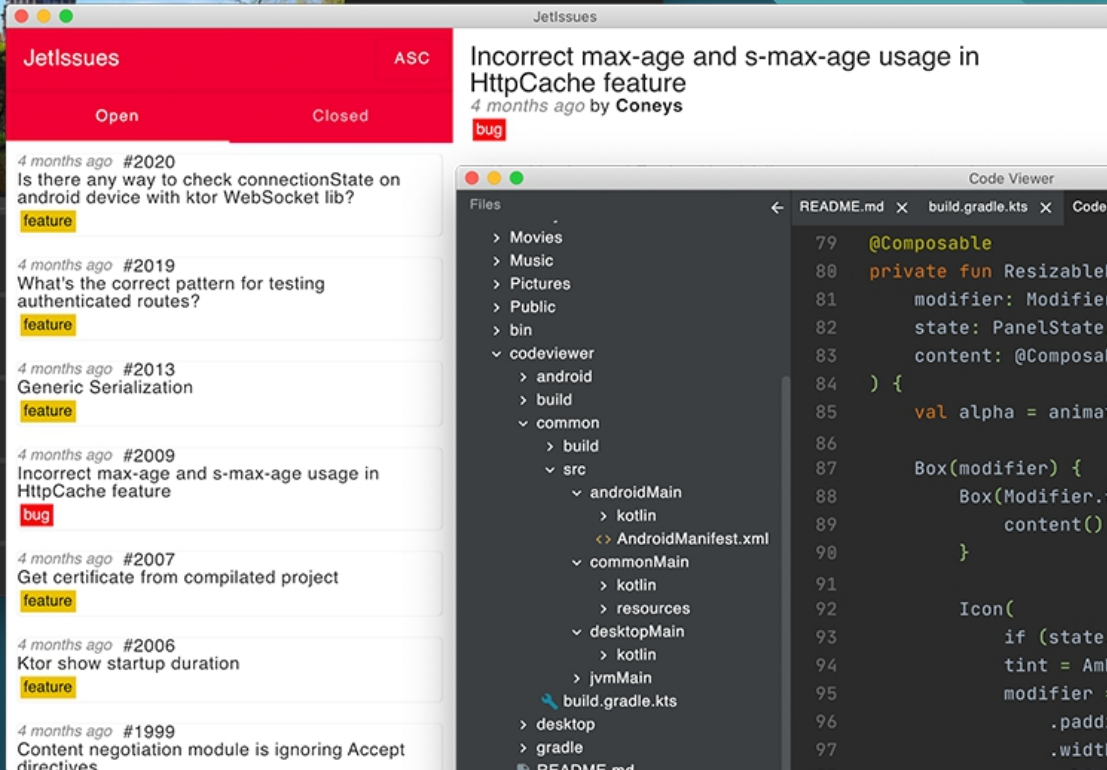
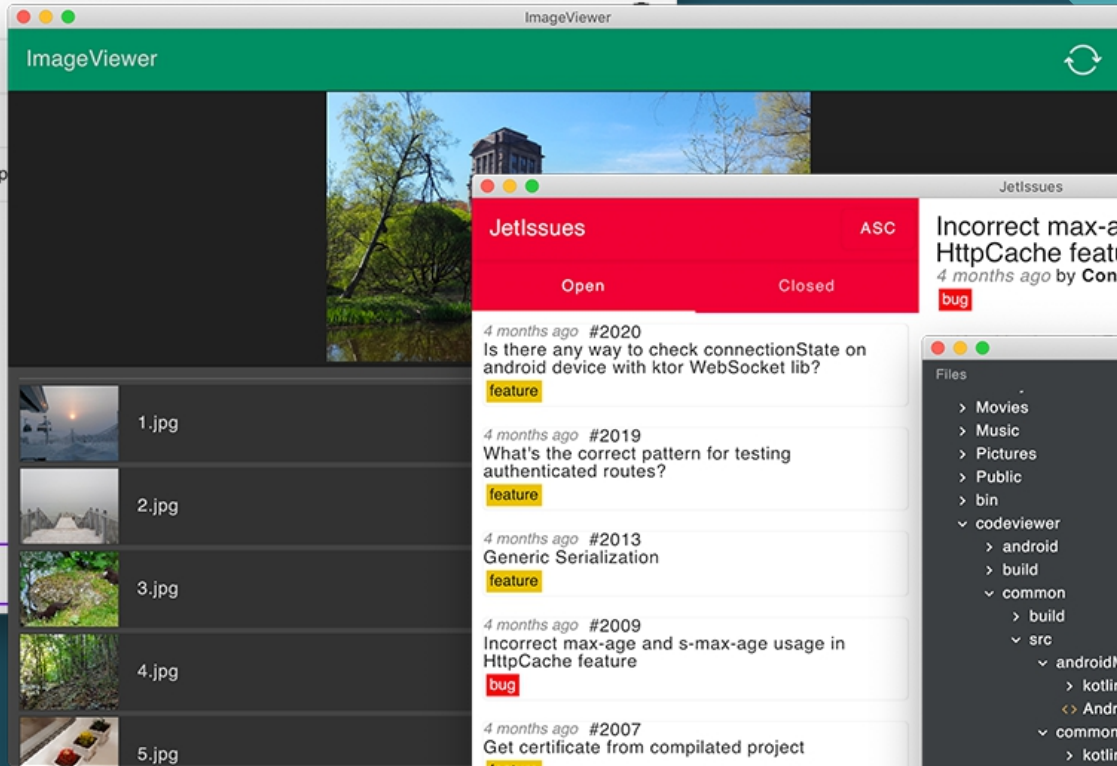
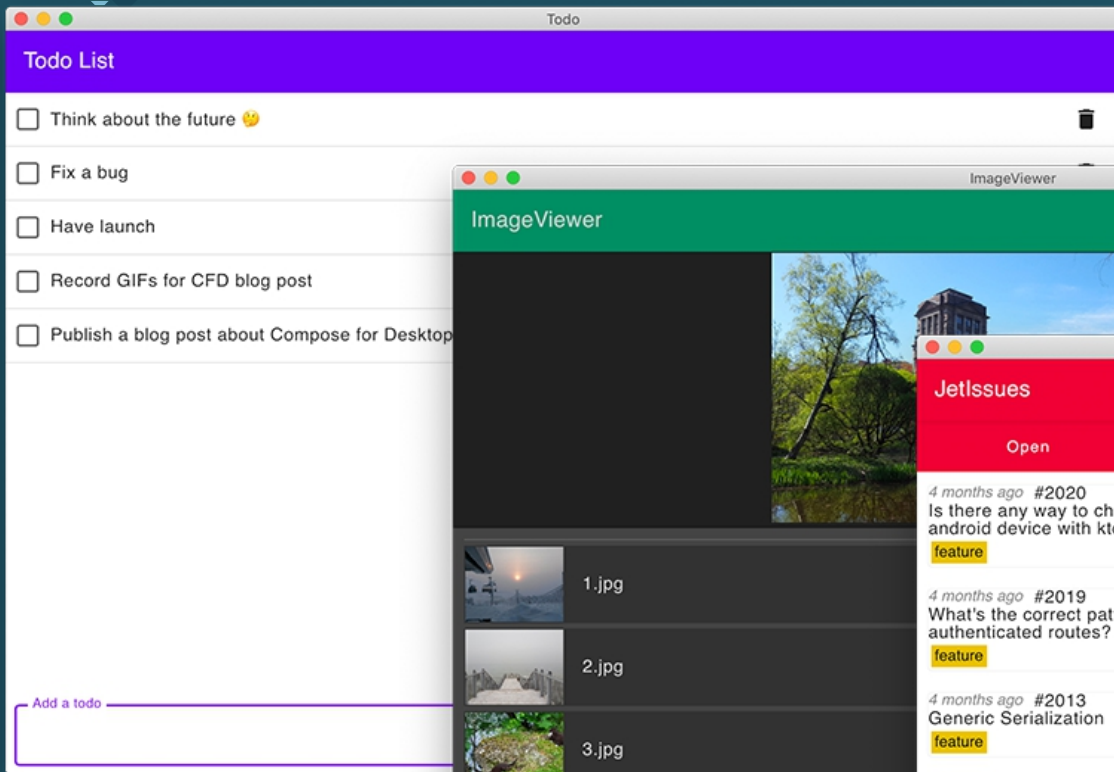
<https://www.jetbrains.com/lp/compose-mpp/>

```
@Composable
fun MessageCard(msg: Message) {
    Row(modifier = Modifier.padding(all = 8.dp)) {
        Image(
            painter = painterResource(R.drawable.profile_picture)
            contentDescription = null,
            modifier = Modifier
                .size(40.dp)
                .clip(CircleShape)
                .border(1.5.dp, MaterialTheme.colors.primary)
        )
        Spacer(modifier = Modifier.width(8.dp))

        // We keep track if the message is expanded
        // variable
        var isExpanded by remember { mutableStateOf(false) }
        // surfaceColor will be updated when the message is expanded
        val surfaceColor: Color by animateColorAsState(
            if (isExpanded) MaterialTheme.colors.primary
            else MaterialTheme.colors.surface
        )

        // We toggle the isExpanded variable when the message is clicked
        Column(modifier = Modifier.clickable { isExpanded = !isExpanded }) {
            Text(
                text = msg.body,
                color = surfaceColor,
                style = TextStyle(
                    color = surfaceColor,
                    background = surfaceColor,
                    padding = 10.dp
                )
            )
        }
    }
}
```





“

"Tenemos que cambiar la tradicional actitud ante la construcción de software. En vez de pensar que nuestra principal tarea es indicar a un ordenador qué hacer, concentrémonos en explicar a las personas lo que queremos que el ordenador haga"

Donald E. Knuth

”

Recursos

- Twitter: <https://twitter.com/joseluisgonsan>
- GitHub: <https://github.com/joseluisgs>
- Web: <https://joseluisgs.github.io>
- Discord: <https://discord.gg/eFMKxfPY>
- Aula Virtual: <https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=247>



Gracias

José Luis González Sánchez

