Interfaces Gráficas

Tipos de Interfaces de Usuario

CLI (Command Line Interface): interfaz gráfica de línea de comandos

Interacción mediante texto. Se basa en el uso de un lenguaje codificado.
Interfaz donde se ingresan comandos de texto para interactuar con una aplicación. Es eficiente pero requiere aprender comandos y su sintaxis.

GUI (Graphical User Interface): interfaz gráfica de usuario

Interacción a través de elementos gráficos. Posibilita una interacción amigable e intuitiva. Permite interactuar con la aplicación haciendo clic y arrastrando objetos. Es más intuitiva pero requiere más recursos y es menos flexible para tareas avanzadas.

Tipos de interfaces gráficas:
- Terminales de texto y programas con menús.
- Terminales de texto y programación secuencial.
- Terminales WIMP (Windows, Icons, Menus, Pointer) y programación por eventos.

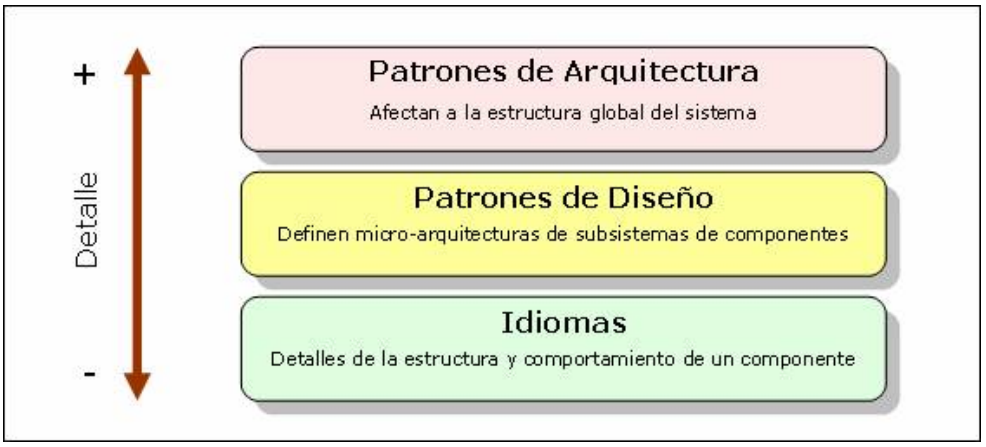
Eventos

- Un evento es un objeto que representa algo interesante que ocurre en un sistema.
- Son previstos pero no planeados (se conoce que ocurrirán pero no cuándo ni las circunstancias).
- Las aplicaciones arrancan desde una cola de eventos y luego espera nuevos eventos.
- Los eventos se procesan en orden, desde la cola de eventos.
- Los eventos pueden ser provocados por el usuario o por el sistema

Eventos de Usuario	Eventos del sistema
Clic del mouse	Intervalo del reloj
Pulsación de una tecla	Cierre del sistema
Movimiento del mouse	
Pérdida de foco	

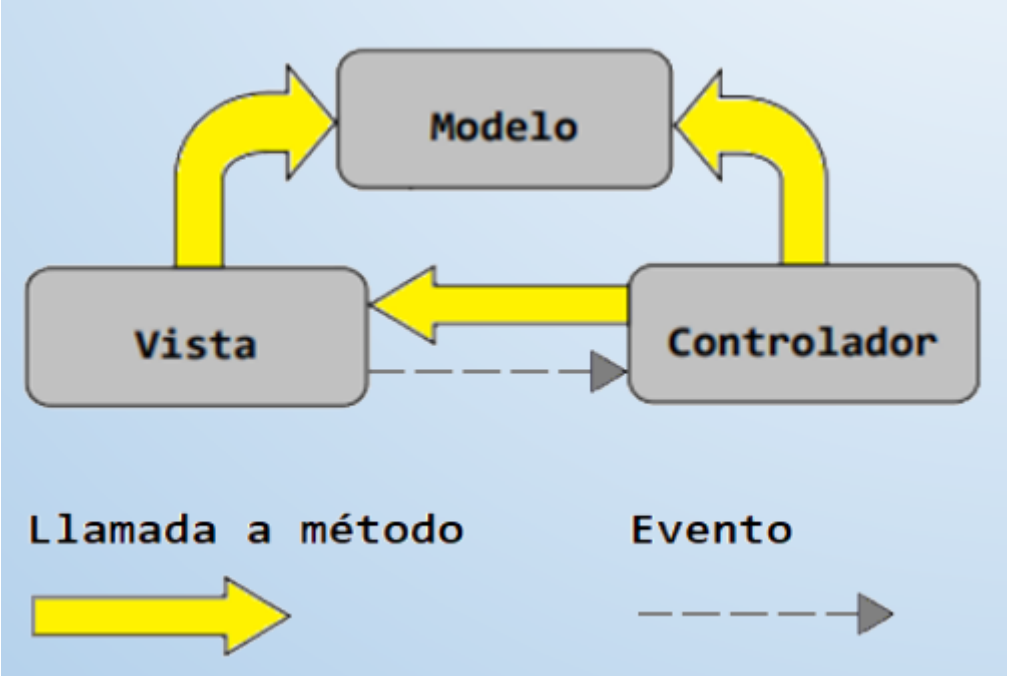
Patrones de Diseño vs Patrones de Arquitectura:

Los patrones de arquitectura expresan un esquema organizativo estructural fundamental para sistemas de software mientras que los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.



Patrón de diseño: relaciones entre clases
Patrón de arquitectura: relación entre paquetes

MVC (Model-View-Controller)
Modelo - Vista - Controlador



- Es un patrón de arquitectura, las responsabilidades están divididas.
- Se puede implementar de más de una forma.

Ejemplo de implementación:

Los datos con que opera el sistema y las operaciones para procesarlos se implementan en el **modelo**.

El usuario interactúa con la **vista**, la cual provee getters y setters para acceder a los contenidos de sus componentes, incluyendo métodos para registrar los objetos listeners,

que en esta arquitectura son parte del **controlador**, ya que es éste quien responde a los eventos que se producen en la vista, accediendo a esta o al modelo según corresponda (mediante llamadas a métodos).

La comunicación entre vista y controlador es a través de eventos (porque la vista no sabe que existe un controlador).

```java
package algo3.hellomundo;

import javafx.application.Application;
import javafx.stage.Stage;

public class App extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Modelo modelo = new Modelo();
        Vista vista = new Vista(stage, modelo);
        Controlador controlador = new Controlador(modelo, vista);
        controlador.iniciar();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

1. Instancio el modelo que quiero.
2. Instancio y la vista y le paso al constructor el modelo (desde cualquier método de la vista puedo acceder al modelo).
3. Instanciar el controlador (le paso el modelo y la vista).

Relación de agregación

- La vista tiene un modelo
- El controlador tiene un modelo y una vista.

```java
package algo3.hellomundo;

import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class Controlador {

    private Modelo modelo;
    private Vista vista;

    public Controlador(Modelo modelo, Vista vista) {
        this.modelo = modelo;
        this.vista = vista;
    }

    public void iniciar() {
        vista.registrarEscucha(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                String nombre = vista.obtenerNombre();
                modelo.actualizarNombre(nombre);
                String saludo = modelo.generarSaludo();
                vista.mostrarSaludo(saludo);
            }
        });
    }
}
```

El controlador recibe el modelo y la vista. El método iniciar() genera el objeto Listener y se lo pasa a la vista. La vista recibe el Listener y se lo aplica a alguno de sus componentes.

Observamos que el método que llama al modelo o mejor dicho la secuencia que se realiza una vez sucedido el evento, se encuentra en controlador, esto no cumple MVC.

```java
package algo3.hellomundo;

public class Modelo {

    private String nombre;
    private String tituloDeApp;
    private String descripcionDeAccion;

    public Modelo() {
        tituloDeApp = "MVC Demo";
        descripcionDeAccion = "Saludar";
    }

    public String getTituloDeApp() {
        return tituloDeApp;
    }

    public String getDescripcionDeAccion() {
        return descripcionDeAccion;
    }

    public void actualizarNombre(String nombre) {
        this.nombre = nombre;
    }

    public String generarSaludo() {
        return "Hola, " + nombre + "!";
    }
}
```

```java
public class Vista {
    private TextField campoNombre;
    private TextField campoSaludo;
    private Button botonSaludar;
    private FlowPane gestorDeLayout;
    private Scene escenaPrincipal;

    public Vista(Stage escenario, Modelo modelo) {
        campoNombre = new TextField();
        campoSaludo = new TextField();
        botonSaludar = new Button(modelo.getDescripcionDeAccion());
        gestorDeLayout = new FlowPane();
        gestorDeLayout.getChildren().add(campoNombre);
        gestorDeLayout.getChildren().add(botonSaludar);
        gestorDeLayout.getChildren().add(campoSaludo);
        escenaPrincipal = new Scene(gestorDeLayout);
        escenario.setScene(escenaPrincipal);
        escenario.setWidth(400);
        escenario.setHeight(120);
        escenario.setTitle(modelo.getTituloDeApp());
        escenario.setResizable(false);
        escenario.show();
    }

    public void registrarEscucha(EventHandler<ActionEvent> eventHandler) {
        botonSaludar.setOnAction(eventHandler);
    }

    public String obtenerNombre() {
        return campoNombre.getText();
    }

    public void mostrarSaludo(String saludo) {
        campoSaludo.setText(saludo);
    }
}
```