Lienzo: Problemas típicos de GUI y “cartas” que JavaFX trae

> Objetivo: tener un mapa mental operativo para construir ejemplos/feature en JavaFX sin reinventar la rueda.

---

1) Concurrencia y el hilo de UI (JavaFX Application Thread)

Problema típico - Haces una operación lenta (HTTP, BD, parseo grande) y la UI se “congela”. - Intentas actualizar UI desde un hilo distinto y aparecen errores o comportamientos raros.

Qué contempla JavaFX - `Platform.runLater(Runnable)` → ejecutar una actualización en el hilo de UI. - `Task<V>` / `Service<V>` → correr trabajo pesado en background + publicar progreso/resultado con propiedades.

Metodología recomendada - Regla: **cálculo pesado fuera del hilo de UI, cambios visuales dentro**. - Usa `Task` cuando es “una corrida” (una operación). Usa `Service` cuando es “repetible” (misma operación varias veces con reinicios).

Micro-sentencia - `Platform.runLater(() -> label.setText("Listo"));`

Ejemplo mínimo - Descargar algo en background: - `Task<String> t = new Task<>() { protected String call(){ ... } };` - `t.setOnSucceeded(e -> label.setText(t.getValue()));` - `new Thread(t).start();`

---

2) Sincronía UI ↔ Datos (Properties, Observable, Bindings)

Problema típico - Mucho `if/else` para habilitar botones, validar campos, actualizar textos.

Qué contempla JavaFX - `StringProperty`, `BooleanProperty`, etc. - `Bindings` (expresiones reactivas) - `ObservableList` (listas reactivas para tablas/listas)

Metodología recomendada - Crear un “estado” (modelo o viewmodel) con `Properties`. - Enlazar UI: la UI *observa*, no “pregunta cada rato”.

Micro-sentencias - `label.textProperty().bind(nombreProperty);` - `btn.disableProperty().bind(Bindings.isEmpty(txt.textProperty()));`

Ejemplo mínimo - Deshabilitar “Guardar” si falta algo: - `guardar.disableProperty().bind(txtNombre.textProperty().isEmpty().or(txtEmail.textProperty().is`

---

3) Eventos: captura, burbujeo, y control de propagación

Problema típico - Un click “se lo come” un control. - Quieres escuchar eventos globales (teclado, mouse) sin pegar listeners por todo.

Qué contempla JavaFX - Fase **capture** (filtro) y **bubble** (handler) - `addEventFilter(...)` (captura) - `addEventHandler(...)` (burbujeo) - `event.consume()`

Metodología recomendada - Eventos **locales**: `setOnAction`, `setOnMouseClicked` en el control. - Eventos **globales**: listener en `Scene` o en el root con filtros/handlers.

Micro-sentencias - `node.addEventFilter(MouseEvent.MOUSE_PRESSED, e -> { ... });` - `node.addEventHandler(KeyEvent.KEY_PRESSED, e -> { ... });`

Ejemplo mínimo - Escape para cerrar: - `scene.addEventHandler(KeyEvent.KEY_PRESSED, e -> { if (e.getCode()==KeyCode.ESCAPE) stage.close(); });`

---

4) Layout responsive (tamaño, reflow, panes)

Problema típico - Al redimensionar la ventana, todo se descuadra. - Con `Pane` (layout manual) todo requiere coordenadas.

Qué contempla JavaFX - Panes de layout: `VBox/HBox`, `BorderPane`, `GridPane`, `StackPane`, `AnchorPane`. - `hgrow/vgrow` y constraints. - Bindings para tamaños.

Metodología recomendada - Para UI “seria”: layouts primero, coordenadas manuales solo para dibujo. - `BorderPane` para estructura general; `VBox/HBox` para barras; `GridPane` para forms.

Micro-sentencias - `HBox.setHgrow(node, Priority.ALWAYS);` - `grid.addRow(0, label, field);`

Ejemplo mínimo - Barra superior + contenido: - `BorderPane root = new BorderPane(); root.setTop(toolbar); root.setCenter(content);`

---

5) Separación de UI y lógica (FXML, Controller, MVVM)

Problema típico - Clases gigantes con UI + lógica mezclada.

Qué contempla JavaFX - `FXML` + `FXMLLoader` - Controllers y `@FXML` - Patrón MVVM con `Properties` (no “oficial”, pero encaja perfecto)

Metodología recomendada - UI declarativa (FXML) + Controller delgado. - Lógica de estado en un “ViewModel” con `Properties`.

Micro-sentencias - `Parent root = FXMLLoader.load(getClass().getResource("/view.fxml"));`

Ejemplo mínimo - Controller usa `textProperty().bind(viewModel.statusProperty())`.

---

6) Navegación entre pantallas (tu Navigator + Scene nuevas)

Problema típico - "Cambiar de ventana" sin duplicar Stage/Scene o sin acoplar todo.

Qué contempla JavaFX - Cambiar `Scene` en el `Stage`. - Alternativa: `scene.setRoot(...)` (si decides Scene persistente).

Metodología recomendada (con tu Navigator actual) - Cada pantalla retorna su `Parent`. - `Navigator.show(view, spec)` crea la `Scene` nueva.

Micro-sentencia - `Navigator.show(new LoginView().view(), WindowSpec.MEDIUM);`

Ejemplo mínimo - Botón navega: - `btn.setOnAction(e -> Navigator.show(new HomeView().view()));`

---

7) Ventanas secundarias, modales y diálogos

Problema típico - Necesitas confirmación, input, o popups sin crear pantallas completas.

Qué contempla JavaFX - `Alert`, `Dialog`, `TextInputDialog` - `Stage` secundario para ventana compleja.

Metodología recomendada - `Alert` para mensajes rápidos. - `Dialog` cuando necesitas "resultado". - `Stage` nuevo para herramientas o paneles grandes.

Micro-sentencias - `new Alert(Alert.AlertType.INFORMATION, "Hola").showAndWait();` - `Optional<String> r = new TextInputDialog().showAndWait();`

---

8) CSS, temas y consistencia visual

Problema típico - UI inconsistente, estilos repetidos en código.

Qué contempla JavaFX - CSS con `scene.getStylesheets()` - `styleClass` y selectores - `PseudoClass` (estados: error/ok)

Metodología recomendada - Estilos globales en CSS; `setStyle(...)` solo para prototipos. - Para validaciones, aplica una clase/pseudoclase.

Micro-sentencias - `scene.getStylesheets().add(getClass().getResource("/app.css").toExternalForm());` - `node.getStyleClass().add("danger");`

---

9) Validación y filtrado de input (TextFormatter)

Problema típico - Evitar letras en campos numéricos, limitar longitud, etc.

Qué contempla JavaFX - `TextFormatter` con filtro

Metodología recomendada - Valida “a la entrada” (filtro) + “a la salida” (reglas).

Micro-sentencia - `txt.setTextFormatter(new TextFormatter<>(change -> change.getControlNewText().matches("\\d*") ? change : null));`

---

10) Teclado, foco y atajos globales (Accelerators)

Problema típico - Ctrl+S para guardar, Ctrl+F buscar, etc.

Qué contempla JavaFX - `scene.getAccelerators().put(KeyCombination, Runnable)` - `requestFocus()`

Metodología recomendada - Atajos globales en la `Scene`. - Foco controlado al entrar a pantalla.

Micro-sentencia - `scene.getAccelerators().put(new KeyCodeCombination(KeyCode.S, KeyCombination.CONTROL_DOWN), this::save);`

---

11) Animaciones y feedback visual

Problema típico - Necesitas transiciones suaves (hover, mostrar panel, loader)

Qué contempla JavaFX - `Timeline`, `KeyFrame` - `FadeTransition`, `TranslateTransition`, etc.

Micro-sentencia - `new FadeTransition(Duration.millis(200), node).play();`

---

12) Dibujo “real” y performance (Canvas vs miles de Nodes)

Problema típico - Dibujar muchas cosas con `Circle/Line/...` como nodos puede volverse pesado.

Qué contempla JavaFX - `Canvas` + `GraphicsContext` para dibujar en una sola superficie.

Metodología recomendada - Si es “pocos elementos interactivos”: `Shapes` como nodos. - Si es “muchísimo dibujo”: `Canvas`.

Micro-sentencia - `GraphicsContext g = canvas.getGraphicsContext2D(); g.fillRect(...);`

---

13) Popups, tooltips, menus contextuales

Problema típico - Click derecho, menús rápidos, ayuda contextual.

Qué contempla JavaFX - `ContextMenu`, `MenuBar`, `Tooltip`, `Popup`

Micro-sentencia - `control.setContextMenu(menu);`

---

14) Fugas de memoria por listeners (muy típico al “navegar”)

Problema típico - Si una pantalla registra listeners a algo global (o a un servicio) y no los remueve, esa pantalla nunca se libera.

Qué contempla JavaFX - `WeakChangeListener` / `WeakInvalidationListener` - (y buenas prácticas) remover listeners al desmontar.

Metodología recomendada - Si registras listeners globales: guarda referencia y remueve en un “dispose()”. - En pantallas simples: preferir listeners ligados a nodos locales.

Micro-sentencia - `property.addListener(new WeakChangeListener<>(listener));`

Guía rápida para tu laboratorio (reglas “de oro”)

1) **Ejemplo = retorna Parent**. No crea `Stage` ni `Scene` (lo hace Navigator). 2) **Eventos locales** (botones/controles): `setOnAction`, `setOnMouse...` en el control. 3) **Eventos globales** (atajos/teclas/mouse global): configúralos en la `Scene` (en Navigator) o con `root.sceneProperty()`. 4) **UI con layout**: usa `BorderPane/VBox/GridPane`; **dibujo**: `Pane + Shapes` o `Canvas`. 5) **No congelar UI**: lo lento en `Task/Service`, UI con `Platform.runLater` o `onSucceeded`. 6) **Evita leaks** al navegar: cuidado con listeners a cosas globales.