EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO
NACIONAL DE MÉXICO

DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMA

UNIDAD 1

E1. INVESTIGACIÓN FLUTTER Y DART

INGENIERÍA EN SISTEMAS COMPUTACIONALES

SEANY CAMPOS CORTÉS 21690072

¿Cómo Dart y el framework Flutter utilizan la POO para crear interfaces gráficas y aplicaciones móviles escalables?

Dart es un lenguaje de programación orientado a objetos, lo que implica que cada elemento, desde funciones hasta valores numéricos, es considerado un objeto. Flutter, al estar basado en Dart, utiliza este enfoque para estructurar sus componentes mediante clases y objetos, haciendo que cada elemento de la interfaz se presente como un widget, es decir, una instancia de una clase. Gracias a principios de POO como la **encapsulación, el polimorfismo y la herencia**, se consigue organizar la lógica de la interfaz de forma clara y eficiente.

POO en Flutter, potencia la reutilización del código y mejora la mantenibilidad de las aplicaciones. Los desarrolladores pueden crear clases base para componentes comunes y extenderlas para generar variaciones sin duplicar código, lo que favorece una **estructura jerárquica y ordenada.** Además, la composición de widgets permite implementar cambios en módulos específicos sin afectar el conjunto del proyecto, contribuyendo significativamente a la escalabilidad de la aplicación.

¿Cuáles son las mejores prácticas para organizar el código en una aplicación Flutter utilizando principios de POO?

Una de las mejores prácticas en Flutter es seguir el **principio de separación de responsabilidades**, dividiendo la aplicación en diferentes capas como presentación, lógica de negocio y datos. Para lograrlo, se recomienda utilizar patrones arquitectónicos como MVVM (Model-View-ViewModel), que permite desacoplar la interfaz de usuario de la lógica de negocio, facilitando la escalabilidad y mantenibilidad del código. **Organizar el código en carpetas bien definidas**, como models, services, widgets y screens, ayuda a mejorar la legibilidad y el mantenimiento del proyecto.

Otro aspecto importante es el **uso adecuado de la encapsulación y la modularidad**. Es recomendable definir clases para representar entidades del dominio de la aplicación y utilizar la herencia o la composición para extender su funcionalidad cuando sea necesario. Además, l**a inmutabilidad de los objetos** y el uso de final o const para datos que no deben cambiar mejoran el rendimiento y evitan errores inesperados. Por último, **documentar el código** mediante comentarios y asegurarse de escribir pruebas unitarias y de integración para garantizar su correcto funcionamiento.

¿Cómo es el uso de clases y herencia en la creación de widgets personalizados?

En Flutter, los widgets son clases que extienden de **StatelessWidget** o **StatefulWidget**, permitiendo crear componentes reutilizables con propiedades personalizadas. **La herencia** se usa cuando se requiere extender la funcionalidad de un widget base para adaptarlo a necesidades específicas. Por ejemplo, un desarrollador puede crear un botón personalizado, añadiendo nuevas propiedades o estilos. Esto permite reutilizar el código y mantener una estructura clara y organizada.
Ejemplo de un widget personalizado:

```dart
// Definición de un widget personalizado llamado CustomButton
class CustomButton extends StatelessWidget {
  // Propiedad para almacenar el texto del botón
  final String text;
  // Propiedad para manejar la acción al presionar el botón
  final VoidCallback onPressed;

  // Constructor que requiere el texto y la acción del botón
  const CustomButton({required this.text, required this.onPressed});

  @override
  Widget build(BuildContext context) {
    // Retorna un botón elevado con las propiedades definidas
    return ElevatedButton(
      onPressed: onPressed, // Asigna la acción al botón
      child: Text(text), // Muestra el texto dentro del botón
    );
  }
}
```

Bibliografía

Introduction to Dart. (s. f.). Dart.dev. Recuperado de https://dart.dev/language

Widget catalog. (s. f.). Flutter.dev. Recuperado de https://docs.flutter.dev/ui/widgets

Dart Team. (n.d.). Effective Dart. Recuperado de https://dart.dev/guides/language/effective-dart

Flutter Team. (n.d.). Layout. Recuperado de https://flutter.dev/docs/development/ui/layout