Buena pregunta. En el proyecto `adminkiosko`, los adaptadores como `RepositorioDeClientesMemoria` y `RepositorioProductosMemoria` implementan interfaces para manejar los datos de clientes y productos en memoria (probablemente para pruebas o como almacenamiento temporal).

Vamos a ver cómo **inyectarlos con GetIt** para desacoplar su uso de la implementación concreta.

---

## 🎯 Paso 1: Identificar las interfaces

Supongamos que tenés algo así:

```
dart
CopiarEditar
abstract class RepositorioDeClientes {
  Future<List<Cliente>> obtenerClientes();
}

class RepositorioDeClientesMemoria implements RepositorioDeClientes {
  // implementación en memoria
}
```

Igualmente para productos:

```
dart
CopiarEditar
abstract class RepositorioDeProductos {
  Future<List<Producto>> obtenerProductos();
}

class RepositorioProductosMemoria implements RepositorioDeProductos {
  // implementación en memoria
}
```

---

## 🧩 Paso 2: Registrar las implementaciones en `service_locator.dart`

```
dart
CopiarEditar
import 'package:get_it/get_it.dart';

import 'package:adminkiosko/adaptadores/repositorio_clientes_memoria.dart';
import 'package:adminkiosko/adaptadores/repositorio_productos_memoria.dart';
import 'package:adminkiosko/dominio/repositorios/repositorio_de_clientes.dart';
import 'package:adminkiosko/dominio/repositorios/repositorio_de_productos.dart';

final getIt = GetIt.instance;

void setupLocator() {
  getIt.registerLazySingleton<RepositorioDeClientes>(
    () => RepositorioDeClientesMemoria(),
  );

  getIt.registerLazySingleton<RepositorioDeProductos>(
    () => RepositorioProductosMemoria(),
  );
}
```

---

## 🧪 Paso 3: Usar los repositorios desde cualquier parte

En una clase o widget donde necesités acceder al repositorio, lo inyectás así:

```
dart
CopiarEditar
import 'package:adminkiosko/dominio/repositorios/repositorio_de_clientes.dart';
import 'package:adminkiosko/service_locator.dart';

class ClienteController {
  final RepositorioDeClientes _repositorio = getIt<RepositorioDeClientes>();

  Future<void> cargarClientes() async {
    final clientes = await _repositorio.obtenerClientes();
    // ...
  }
}
```

Esto te permite:

- Usar implementaciones en memoria (`Memoria`) en desarrollo/test.
- Sustituir fácilmente por implementaciones reales (`Firestore`, `SQLite`, etc.) sin cambiar el resto del código.

---

## 🧪 Bonus: cambiar implementación para producción

Supongamos que en producción querés usar Firestore:

```
dart
CopiarEditar
void setupLocator() {
  if (kReleaseMode) {
    getIt.registerLazySingleton<RepositorioDeClientes>(
      () => RepositorioDeClientesFirestore(),
    );
  } else {
    getIt.registerLazySingleton<RepositorioDeClientes>(
      () => RepositorioDeClientesMemoria(),
    );
  }
}
```