Patrones de Diseno - Challup

BugBusters

25 de octubre de 2025

1. Singleton

1.1. Proposito

Garantizar que una clase tenga una unica instancia y proporcionar un punto de acceso global a ella.

1.2. Implementacion en el Proyecto

Se utiliza en la clase `DatabaseHelper` para gestionar la conexion a la base de datos SQLite.

1.3. Beneficios

- **Control de recursos**: Evita multiples conexiones a la base de datos, optimizando el uso de recursos del sistema.
- **Acceso consistente**: Proporciona un punto unico de acceso a la base de datos en toda la aplicacion.
- **Gestion centralizada**: Facilita el mantenimiento y la administracion de la conexion a la base de datos.

1.4. Codigo de Ejemplo

```
class DatabaseHelper {
  // Instancia unica de la clase
  static final DatabaseHelper instance = DatabaseHelper._init();
  static Database? _database;

  // Constructor privado que evita la instanciacion directa
  DatabaseHelper._init();

  // Metodo para obtener la base de datos
  Future<Database> get database async {
    if (_database != null) return _database!;
```

```
    _database = await _initDB('challup.db');
    return _database!;
  }
}
```

Listing 1: Implementacion Singleton en DatabaseHelper

1.5. Uso en la Aplicacion

```
// En cualquier parte de la aplicacion
final db = await DatabaseHelper.instance.database;
```

Listing 2: Uso del Singleton en la aplicacion

2. Factory Method

2.1. Proposito

Definir una interfaz para crear un objeto, pero dejar que las subclases decidan que clase instanciar. Permite que una clase delegue la instanciacion a las subclases.

2.2. Caso de Uso Propuesto

En la aplicacion Challup, existen diferentes tipos de retos (diarios, globales, personalizados). El patron Factory Method seria ideal para crear estos diferentes tipos de retos sin acoplar el codigo cliente a clases concretas.

2.3. Beneficios

- **Flexibilidad**: Facilita la adicion de nuevos tipos de retos sin modificar el codigo existente.
- **Desacoplamiento**: Separa la logica de creacion de objetos del codigo que los utiliza.
- **Mantenibilidad**: Cambios en la creacion de objetos no afectan al codigo cliente.

2.4. Ejemplo Conceptual

```
// Interfaz base para todos los retos
abstract class Reto {
  String descripcion;
  int puntos;

  Reto(this.descripcion, this.puntos);

  void completar();
}

// Implementaciones concretas
```

```
class RetoDiario extends Reto {
  DateTime fecha;

  RetoDiario(String descripcion, int puntos, this.fecha)
    : super(descripcion, puntos);

  @override
  void completar() {
    // Logica especifica para completar un reto diario
  }
}

class RetoGlobal extends Reto {
  int creadorId;
  DateTime fechaPublicacion;

  RetoGlobal(String descripcion, int puntos, this.creadorId, this.
   fechaPublicacion)
    : super(descripcion, puntos);

  @override
  void completar() {
    // Logica especifica para completar un reto global
  }
}

// Fabrica abstracta
abstract class FabricaReto {
  Reto crearReto(String tipo, String descripcion, int puntos, Map<String,
   dynamic> parametros);
}

// Implementacion concreta de la fabrica
class FabricaRetoConcreta implements FabricaReto {
  @override
  Reto crearReto(String tipo, String descripcion, int puntos, Map<String,
   dynamic> parametros) {
    switch (tipo) {
      case 'diario':
        return RetoDiario(descripcion, puntos, parametros['fecha']);
      case 'global':
        return RetoGlobal(descripcion, puntos, parametros['creadorId'],
   parametros['fechaPublicacion']);
      default:
        throw Exception('Tipo de reto no soportado: $tipo');
    }
  }
}
```

Listing 3: Ejemplo de Factory Method

2.5. Uso en la Aplicacion

```
// Crear una fabrica
final fabrica = FabricaRetoConcreta();

// Crear diferentes tipos de retos usando la fabrica
final retoDiario = fabrica.crearReto(
  'diario',
  'Caminar 10,000 pasos',
  50,
  {'fecha': DateTime.now()}
);

final retoGlobal = fabrica.crearReto(
  'global',
  'Plantar un arbol',
  200,
  {
    'creadorId': 123,
    'fechaPublicacion': DateTime.now()
  }
);
```

Listing 4: Uso del Factory Method

2.6. Escenario de Aplicacion en Challup

Cuando un usuario solicita un nuevo reto, el sistema puede utilizar una fabrica para crear el tipo de reto apropiado basado en criterios como:

- Tipo de reto (diario, semanal, global)
- Categoria del reto (salud, educacion, medio ambiente)
- Complejidad del reto (facil, medio, dificil)

Esto permitiria una facil expansion del sistema para incluir nuevos tipos de retos en el futuro sin modificar el codigo existente.