Universidad Don Bosco

Trabajo de Investigación

Integrantes

* Castellanos Martinez, Andrea Alejandra
* Mejia Sosa, Marleny Guadalupe

Docente:

* Alexander Sigüenza

lunes 14 junio 2021, Soyapango

Arquitectura Clean

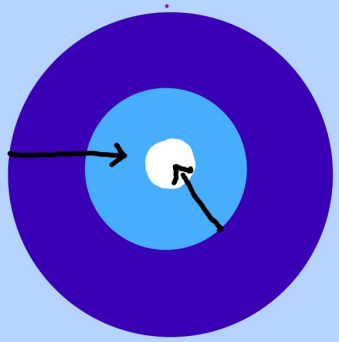
Principios cuya finalidad es poder ocultar los detalles de Implementación a la logica de dominio de la aplicación

El codigo se estructura en capas contiguas, esto quiere decir que solo se tiene comunicación en las Capas que se encuentran inmediatamente al lado

Caracteristica

Regla de dependencia

Las dependencias solo pueden apuntar hacia adentro.



"Un circulo interior nunca debe conocer nada sobre un circulo exterior, los circulos exteriores sí pueden conocer los circulos interiores"

Niveles

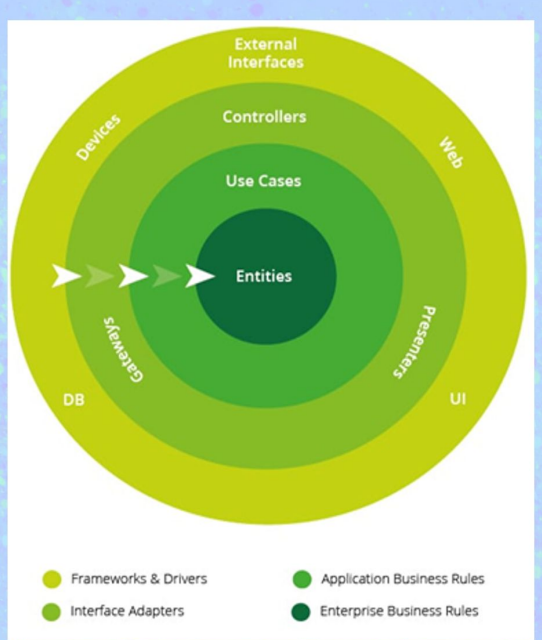
La arquitectura se compone por las siguientes capas:

- UI: Interfaz de usuario (html, xml, etc)
- Presenters: Clases que se suscriben a los eventos generados por la interfaz

Use Cases: evaluación de reglas de negocio y tomas de decisiones

Entities modelo de datos de la aplicación

Comunicación con servicios web, cache de datos



✓ Ventajas

Independencia: Cada capa tiene su modelo arquitectonico.

Estructuración: mejor organización del codigo

Desacoplamiento: Cada capa es independiente de las demas, por lo que podria reemplazar, Reutilizar o usar diferente arquitectura

Facilidad de testeo: Se pueden llevar a cabo testeos de cada capa y de Integración

Metodologia: todo el equipo debe conocer la metodologia que se esta aplicando, asi como conocer las reglas y saber aplicarlas.

Complejidad: la velocidad de desarrollo al principio es menor debido a que hay que establecer estructura y el equipo debe adaptarse

X Desventajas

Propiedades

Solid

5 Principios de Diseño

Se usan para hacer el diseño de Software mas estable, flexible, entendible, mantenibles, escalables y robustos en el futuro.

Los principios de Solid son:

Single Responsability Principle
Open/Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle

Solid Responsability

Se basa en que cada clase o metodo solo debe hacer una cosa, sencilla y concreta.
Si un objeto tiene un solo cometido este sera más facil de mantener.

Open / Closed

Dice que una clase o metodo debe estar abierto a extensiones, pero cerrado a modificaciones.

Liskov Substitution

Los objetos de un desarrollo deberian ser reemplazables por instancias de subtipos sin alterar el correcto funcionamiento del desarrollo
Cualquier subclase deberia poder ser sustituibles por la clase padre

Interface Segregation

Es mejor definir una serie de metodos abstractos a traves de una serie de interfaces para que implementen nuestras clases.
Cada interface tendra una unica responsabilidad.
No se contendrán metodos que no se utilizen

Dependency Inversion

Conseguir desacoplar las clases del desarrollo, que una clase interactue con otras clases sin que la conozca directamente

Ventajas

Reducir la Complejidad del Codigo

Incrementar la extensibilidad y el mantenimiento

Disminuir errores e Implementar Reusabilidad

Reducir el alto acoplamiento

Software mas facil de testear

Patrones de diseño

Solución General y reutilizable aplicable a diferentes problemas de diseño de Software

Se trata de plantillas que identifican Problemas en el sistema y proporcionan Soluciones apropiadas a problemas generales.

Ayudan a estar seguros de la validez del Código, ya que son soluciones que funcionan y han sido aprobadas por muchisimos desarrolladores siendo menos propensos a errores

Tipos de Patrones

Abstract factory

una Interfaz crea Conjuntos o familias de objetos relacionados sin especificar el nombre de la Clase

Builder Patterns

Permite producir diferentes tipos y representaciones de un objeto utilizando el mismo codigo

Patrones Cracionales

proporcionan mecanismos de creación de objetos

Factory Method

Proporciona una interfaz para crear objetos en una Super clase pero permite que las Subclases alteren el tipo de objeto

Prototype

Permite copiar objetos existentes sin hacer que Su codigo dependa de Sus clases

Singleton

Restringe la creacion de Intancias de una clase a un unico objeto.

Patrones estructurales

Facilitan soluciones y estandares eficientes con respecto a las composiciones de clase y las estructuras de objetos.

Adapter

Vincular 2 Interfaces que no son Compatibles y utilizan sus funcionalidades

Bridge

2 clases desarrolladas de manera independiente si hay alteración estructural pueden unirse utilizando una interfaz como puente

Composite

Se usa para agrupar objetos como un solo objeto

Decorator

Restringe la alteración del objeto mientras se le agrega una nueva funcionalidad

Facade

Proporciona una interfaz simplificada para una biblioteca, marco o cualquier otro conjunto complejo de clases

Flyweight

Se usa para reducir el uso de memoria y mejorar el rendimiento al reducir la creación de objetos

Proxy

Se utiliza para crear objetos que pueden representar funciones de otras clases u objetos y la interfaz se utiliza para acceder a estas funcionalidades

Patrones de Comportamiento

Se ocupa de la comunicación entre objetos de clases
Se utiliza para detectar la presencia de patrones de Comunicación ya presentes y pueden manipular estos patrones

Command
Convierte una Solicitud en un objeto Independiente con la información de la Solicitud

Interpreter
Evalua el lenguaje o expresión al crear una Interfaz

Iterator
acceso secuencial a un número de elementos presentes dentro de un Objeto

Mediator
proporciona una Comunicacion facil

Memento
Permite recorrer elementos de una Colección

Chain of responsability
evita acoplar el emisor de una peticion a su receptor dando a más de un objeto la posibilidad de responder

Observer
Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le sucada al objeto.

State
el comportamiento de una clase varía con su estado, representado por el objeto de Contexto

Strategy
definir una familia de algoritmos poner cada uno de ellos en una clase separada y hacer objetos intercambiables

Template Method
Se usa en componentes con similitud donde se puede Implementar una plantilla de Codigo para probarlos

Visitor
definir una nueva operación sin Introducir las modificaciones a una estructura de objeto existente