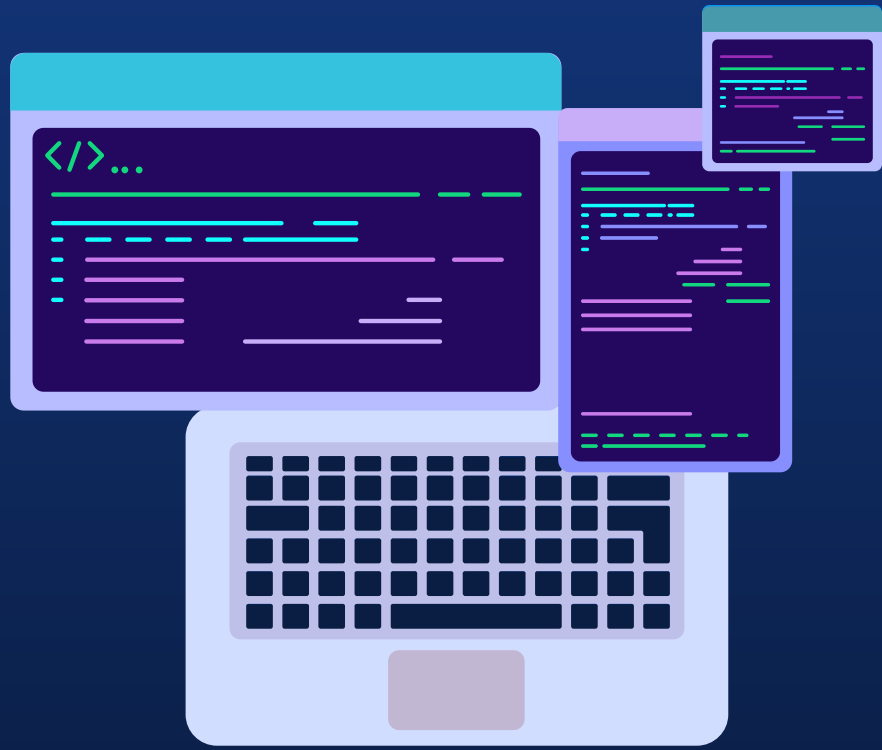
Patrones GRASP



Contenido

- **¿Qué es un Patrón?**
- **¿Por qué usar patrones?**
- Definición de GRASP
- Ventajas y Alcance
- **Experto en información**
- **Creador**
- **Alta cohesión**
- **Bajo acoplamiento**
- **Controlador**
- *Fabricación Pura*
- *Polimorfismo*
- *Indirección*
- *No hables con extraños*

¿Qué es un Patrón?

- Podemos definir un patrón como la solución a un problema dentro de un contexto dado.
- Es recurrente, lo que hace que sea relevante para otras soluciones.
- Es una solución a un problema de diseño no trivial que es *efectiva y reusable*.
- Proporcionan respuestas a un conjunto de problemas similares, siendo una solución para varios contextos.

Por definición

Contexto — Son las situaciones recurrentes a las que es posible aplicar un patrón

Problema — Es el conjunto de metas y restricciones que se dan en ese contexto

Solución — Es el diseño a aplicar para conseguir las metas dentro de las restricciones

¿Por qué usar patrones?

01
Software robusto

Bajo impacto de resistencia al cambio

02
Responsabilidades claras

Ayudan a especificar interfaces

03
Reutilización de código

Creando componentes independientes pero reutilizables

04
Documentación

El uso de patrones favorece la documentación implícita

05
Reducción de errores

Los patrones evitan el cometer errores de diseño comunes

06
Mejora de mantenimiento

Al tener una estructura organizada, el mantenimiento es más sencillo

Patrones

PROBLEM

¿Cómo mejorar la calidad del código, escalabilidad, reutilización y respetar estándares en la industria?

SOLUTION

Uso de Patrones.

GRASP

Patrones de software para la asignación general de responsabilidades

GRASP

General Responsibility Assignment Software Patterns

Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades

- Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto su comportamiento.
- Hay dos tipos de responsabilidades: **Conocer** y **Hacer**

Tipos de responsabilidades

CONOCER

Conocer los datos privados encapsulados

Conocer los objetos relacionados

Conocer las cosas que puede derivar o calcular

HACER

Hacer algo él mismo, como crear un objeto o hacer un cálculo

Iniciar una acción en otros objetos

Controlar y coordinar actividades en otros objetos

Calidad de diseño = Responsabilidades claras

Pensar sobre responsabilidades

Un Software orientado a POO define responsabilidades y postcondiciones de operaciones entre objetos.

GRASP

5 patrones principales

- Experto
- Creador
- Alta cohesión
- Bajo acoplamiento
- Controlador

4 patrones adicionales

- Fabricación pura
- Polimorfismo
- Indirección
- No hables con extraños

Al momento de especificar un patrón definiremos:

- Descripción
- Escenario de uso
- Solución concreta
- Consecuencias
- Patrones Relacionados

01

Experto en información

Experto en información

Problema

¿Quién debería ser el responsable de conocer la información?

Solución

Asignar una responsabilidad al experto. Es la clase que tiene la información necesaria para realizar un acción

Beneficios

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.
- - El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de comprender y de mantener

02

Creador

Creador

Problema

¿quién debería ser el responsable de la creación de una nueva instancia de alguna clase?

Solución

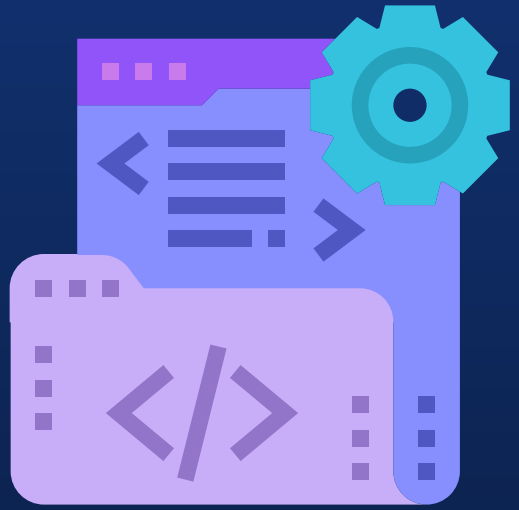
El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos

Beneficios

- El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.
- Al escoger como creador, se da soporte al bajo acoplamiento.

03

Bajo Acoplamiento



Bajo Acoplamiento

Problema

¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización?

Solución

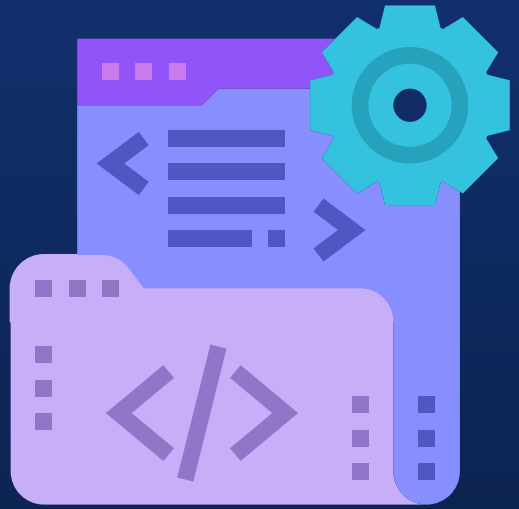
Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.

Bajo Acoplamiento

- El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.
- Acoplamiento bajo significa que una clase no depende de muchas clases.
- Acoplamiento alto significa que una clase recurre a muchas otras clases. Esto presenta los siguientes problemas:
    - Los cambios de las clases afines ocasionan cambios locales.
    - Difíciles de entender cuando están aisladas.
    - Difíciles de reutilizar puesto que dependen de otras clases.

04

Alta Cohesión



Alta Cohesión

Problema — ¿Cómo mantener la complejidad manejable?

Solución — Asignar una responsabilidad de manera que la cohesión permanezca alta.

Alta Cohesión

- En cuanto al diseño de objetos, la cohesión (cohesión funcional) es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento.
- Una clase con baja cohesión hace muchas cosas no relacionadas, o hace demasiado trabajo:
    - Clases difíciles de entender;
    - Difíciles de reutilizar
    - Difíciles de mantener
    - Delicadas, constantemente afectadas por los cambios.
- Como regla empírica, una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidad altamente relacionada, y no realiza mucho trabajo. Colabora con otros objetos para compartir el esfuerzo si la tarea es extensa.
- No es conveniente recargar el trabajo o incluir funcionalidad en la clase que responde al evento del sistema

Alta Cohesión - Ventajas

Incrementa la claridad y facilita la comprensión del diseño

Simplifica el mantenimiento y las mejoras

Soporta a menudo bajo acoplamiento e incrementa la reutilización

05

Controlador

Controlador

Problema

¿Quién debería encargarse de atender un evento del sistema?

Solución

Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase

Controlador - Características

- Normalmente un controlador delega en otros objetos el trabajo que se necesita hacer; coordina o controla la actividad. No realiza mucho trabajo por sí mismo.

- Tipos de controladores:
  - Controlador de Fachada: Representa al sistema global, dispositivo o subsistema.

  - Controlador de casos de uso: Construcción artificial para dar soporte al sistema. Se utilizan cuando los Controladores de Fachada conduce a diseños con baja cohesión o alto acoplamiento.

- El objeto Controlador es normalmente un objeto del lado del cliente en el mismo proceso que la UI.

Controlador - Ventajas

- Aumenta el potencial para reutilizar las interfaces;
- Razonamiento sobre el estado (secuencia de pasos) de los casos de uso.
- Reflexionar sobre el estado del caso de uso. A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente.

06

Fabricación Pura

Fabricación Pura

Problema

¿Cómo proceder cuando las soluciones encontradas comprometen la cohesión y el acoplamiento?

Solución

Asigne un conjunto cohesivo de responsabilidades a una clase artificial (no representa • ningún concepto del dominio del problema)

Beneficios

Las fabricaciones puras suelen ser muy cohesivas y reutilizables

07

Indirección

Fabricación Pura

Problema

¿Dónde asignar responsabilidades para evitar/reducir el acoplamiento directo entre elementos y • mejorar la reutilización?

Solución

Asigne la responsabilidad a un objeto que medie entre los elementos. Ahora el acoplamiento es indirecto

Beneficios

Disminuye el acoplamiento

08

No hables con extraños

Fabricación Pura

Problema

¿A quién asignar las responsabilidades para evitar conocer la estructura de los objetos indirectos?

Solución

Se asigna la responsabilidad a un objeto directo del cliente para que colabore con un objeto indirecto, de modo que el cliente no necesite saber nada del objeto indirecto.

Beneficios

Con esto se busca no acoplar un cliente al conocimiento de objetos indirectos, ni a las representaciones internas de objetos directos.

Los objetos directos son "conocidos" del cliente, los objetos indirectos son “extraños", y un cliente debería tener sólo conocidos, no extraños.