Universidad Tecnológica del Norte de Guanajuato

Organismo Público Descentralizado del Gobierno del Estado

"Educación y progreso para la vida"

Alumno:

Daniel Mendoza Arredondo

Materia:

Programación de aplicaciones

Profesor:

Anastasio Rodríguez García

Tema:

Investigación Solid y Grasp

Grupo:

GITI7083-S

GRASP

En diseño orientado a objetos, GRASP son patrones generales de software para asignación de responsabilidades, es el acrónimo de "GRASP (object-oriented design General Responsibility Assignment Software Patterns)". Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Problema: ¿Cuál es el principio general para asignar responsabilidades a los objetos?

Solución: Asignar una responsabilidad al experto en información.

Beneficios: Se mantiene el encapsulamiento, los objetos utilizan su propia información para llevar a cabo sus tareas. Se distribuye el comportamiento entre las clases que contienen la información requerida. Son más fáciles de entender y mantener.

Controlador

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.

Alta cohesión

Nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.

- Cohesión Coincidente: El módulo realiza múltiples tareas, sin ninguna relación entre ellas.
- Cohesión Lógica: El módulo realiza múltiples tareas relacionadas, pero, en tiempo de ejecución, sólo una de ellas será llevada a cabo.
- Cohesión Temporal: Las tareas llevadas a cabo por un módulo tienen, como única relación "que deben ser ejecutadas al mismo tiempo".
- Cohesión de Procedimiento: La única relación que guardan las tareas de un módulo es que corresponden a una secuencia de pasos propia del “producto”.
- Cohesión de Comunicación: Las tareas corresponden a una secuencia de pasos propia del “producto” y todas afectan a los mismos datos.

- Cohesión de Información: Las tareas llevadas a cabo por un módulo tienen su propio punto de arranque, su codificación independiente y trabajan sobre los mismos datos. El ejemplo típico: OBJETOS
- Cohesión Funcional: Cuando el módulo ejecuta una y sólo una tarea, teniendo un único objetivo a cumplir, se dice que tiene Cohesividad Funcional.

Bajo acoplamiento

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases

1. Acoplamiento de Contenido: Cuando un módulo referencia directamente el contenido de otro módulo. (En lenguajes de alto nivel es muy raro)
2. Acoplamiento Común: Cuando dos módulos acceden (y afectan) a un mismo valor global.
3. Acoplamiento de Control: Cuando un módulo le envía a otro un elemento de control que determina la lógica de ejecución del mismo.