GRASP

General Responsibility Assignment Software Patterns (or Principles), abbreviated GRASP, is a set of *"nine fundamental principles in object design and responsibility assignment"* first published by ***Craig Larman*** in his 1997 book Applying UML and Patterns.

Information expert

What is a basic principle by which to assign responsibilities to objects?

Assign responsibility to the class that has the information needed to fulfill it.

Information expert (also expert or the expert principle) is a principle used to determine where to delegate responsibilities such as methods, computed fields, and so on.

This will lead to placing the responsibility on the class with the most information required to fulfill it.

Related Pattern or Principle: Low Coupling, High Cohesion

Creator

Who creates object A?

In general, Assign class B the responsibility to create object A if one, or preferably more, of the following apply:

- B contain or compositely aggregate instances of A
- B record instances of A
- B closely use instances of A
- B have the initializing information for instances of A and pass it on creation.

Related Pattern or Principle: Low Coupling, Factory pattern

Protected variations

How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

Identify points of predicted variation or instability; assign responsibilities to create a stable interface around them.

High cohesion

High cohesion is an evaluative pattern that attempts to keep objects appropriately focused, manageable and understandable. High cohesion is generally used in support of low coupling. High cohesion means that the responsibilities of a given set of elements are strongly related and highly focused on a rather specific topic. Breaking programs into classes and subsystems, if correctly done, is an example of activities that increase the cohesive properties of named classes and subsystems. Alternatively, low cohesion is a situation in which a set of elements, of e.g., a subsystem, has too many unrelated responsibilities. Subsystems with low cohesion between their constituent elements often suffer from being hard to comprehend, reuse, maintain and change as a whole..

Low coupling

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is an evaluative pattern that dictates how to assign responsibilities for the following benefits:

- lower dependency between the classes,
- change in one class having a lower impact on other classes,
- higher reuse potential.

Pure fabrication

A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived (when a solution presented by the information expert pattern does not). This kind of class is called a "service" in domain-driven design.

Related Patterns and Principles: Low Coupling, High Cohesion.

Polymorphism

How to handle alternatives based on type? How to create pluggable software components?

When related alternatives or behaviors vary by type (class), assign responsibility for the behavior—using polymorphic operations—to the types for which the behavior varies. (Polymorphism has several related meanings. In this context, it means "giving the same name to services in different objects".)

Controller

Who should be responsible for handling an input system event?

A use case controller should be used to deal with all system events of a use case, and may be used for more than one use case. For instance, for the use cases Create User and Delete User, one can have a single class called UserController, instead of two separate use case controllers.

Related Pattern or Principle: Command, Facade, Layers, Pure Fabrication

Indirection

Where to assign responsibility, to avoid direct coupling between two (or more) things? How to de-couple objects so that low coupling is supported and reuse potential remains higher?

Assign the responsibility to an intermediate object to mediate between other components or services so that they are not directly coupled.

The intermediary creates an indirection between the other components.

By **asminog**
cheatography.com/asminog/

Published 29th June, 2022.
Last updated 29th June, 2022.