GRASP (padrão orientado a objetos)

Origem: Wikipédia, a enciclopédia livre.

General responsibility assignment software patterns (ou **principles**), abreviado **GRASP**, consiste em diretrizes para atribuir responsabilidade a classes e objetos em projeto orientado a objetos.

Os diferentes padrões e princípios utilizados no GRASP são: *controller* (controlador), *creator* (criador), *indirection* (indireção), *information expert* (especialista na informação), alta coesão, baixo acoplamento, polimorfismo, *pure fabrication* (fabricação/invenção pura) e *protected variations* (variações protegidas). Todos esses padrões respondem a algum problema, e esses problemas são comuns a quase todos os projetos de desenvolvimento de software. Essas técnicas não foram inventadas a fim de criar novas formas de trabalho, mas para melhor documentar e padronizar os antigos e amplamente praticados princípios de programação em padrões orientado a objetos.

Craig Larman, cientista da computação, afirma que "a ferramenta crucial de projeto para desenvolvimento de software é uma mente bem educada em princípios de projeto. Não é UML ou qualquer outra tecnologia". [1] Assim, GRASP é realmente um conjunto de ferramentas mentais, um auxílio de aprendizagem para ajudar no projeto de software orientado a objetos.

Índice

Padrões

Controller (controlador)

O padrão **controlador** atribui a responsabilidade de manipular eventos do sistema para uma classe que não seja de interface do usuário (UI) que representa o cenário global ou cenário de caso de uso. Um objeto controlador é um objeto de interface não-usuário, responsável por receber ou manipular um evento do sistema.

Um caso de uso controlador deve ser usado para lidar com *todos* os eventos de casos de uso e pode ser usado para mais de um caso de uso (por exemplo, para casos de uso como *Criar usuário* e *Excluir usuário*, pode ter um único *UserController*, em vez de dois casos de uso *controllers* separados).

É definido como o primeiro objeto além da camada UI que recebe e coordena ("controla") operações do sistema. O controlador deve delegar o trabalho que precisa ser feito para outros objetos; ele coordena ou controla a atividade. Ele não deve fazer muito trabalho por si próprio. O *Controller* GRASP pode ser considerado uma parte da camada de aplicação/serviço [2] (assumindo que a aplicação tenha feito uma distinção explícita entre a camada de aplicativo/serviço e a camada de domínio em um sistema orientado a objetos com camadas comuns em uma arquitetura lógica do sistema de informações).

Creator (criador)

A criação de objetos é uma das mais comuns atividades em um sistema orientado a objetos. Descobrir qual classe é responsável por criar objetos é uma propriedade fundamental da relação entre objetos de classes particulares.

Em geral, uma classe `B` deve ser responsável por criar instâncias de classe `A` se uma, ou preferencialmente mais, das seguintes afirmações se aplicam:

- Instâncias de `B` contêm ou agregam instâncias de `A`;
- Instâncias de `B` gravam instâncias de `A`;
- Instâncias de `B` utilizam de perto instâncias de `A`;
- Instâncias de `B` têm as informações de iniciação das instâncias de `A` e passam isso na criação.

Indirection

O padrão de **indireção** suporta baixo acoplamento (e potencial de reutilização) entre dois elementos, atribuindo a objeto intermediário a responsabilidade de ser o mediador entre eles. Um exemplo é a introdução do componente controlador para mediação entre dados (modelo) e sua representação (visualização) no padrão MVC.

Information expert (Especialista na informação)

Especialista na informação é um princípio utilizado para determinar onde delegar responsabilidades. Essas responsabilidades incluem métodos, campos computados, e assim em diante.

Usando o princípio *information expert*, uma abordagem geral para atribuir responsabilidades é olhar para uma determinada responsabilidade, determinar a informação necessária para cumpri-la e depois determinar onde essa informação está armazenada.

Information expert guia colocará a responsabilidade na classe com a maioria das informações necessárias para cumpri-la. [3]

Alta coesão

Alta coesão é um padrão avaliativo que tenta manter os objetos adequadamente focados, gerenciáveis e compreensíveis. A alta coesão é geralmente utilizada em suporte de baixo acoplamento. A alta coesão significa que as responsabilidades de um determinado elemento estão fortemente relacionadas e altamente focadas. A quebra de programas em classes e subsistemas é um exemplo de atividades que aumentam as propriedades coesivas de um sistema. Alternativamente, a baixa coesão é uma situação em que um determinado elemento tem muitas responsabilidades distintas, não relacionadas. Elementos com baixa coesão muitas vezes sofrem de ser difíceis de entender, reutilizar, manter e são avessos à mudança.[4]

Baixo acoplamento

O acoplamento é uma medida de quão forte um elemento está conectado, tem conhecimento ou depende de outros elementos. O **baixo acoplamento** é um padrão de avaliação que determina como atribuir responsabilidades de suporte:

- menor dependência entre as classes,
- mudança em uma classe com menor impacto em outras,
- maior potencial de reutilização.

Polimorfismo

De acordo com o princípio do **polimorfismo**, a responsabilidade de definir a variação dos comportamentos com base no tipo é atribuída ao tipo para o qual essa variação ocorre. Isto é conseguido utilizando operações polimórficas. O usuário do tipo deve usar operações polimórficas em vez de ramificações explícitas com base no tipo.

Pure fabrication (invenção pura)

Uma **fabricação/invenção pura** é uma classe artificial que não representa um conceito no domínio do problema, especialmente feito para conseguir baixo acoplamento, alta coesão e o potencial de reutilização derivado (quando uma solução apresentada pelo padrão *information expert* não é). Esse tipo de classe é chamado de "serviço" em padrão orientado a domínio.

Protected variations (variações protegidas)

O padrão **variações protegidas** protege elementos das variações em outros elementos (objetos, sistemas, subsistemas) envolvendo o foco de instabilidade com uma interface e usando polimorfismo para criar várias implementações desta interface.

Veja também

- Padrão de projeto de software

Notas

1. Larman 2005, p. 272.
2. «Application Layer like business facade?» (https://groups.yahoo.com/neo/groups/domaindrivendesign/conversations/messages/7582). *Yahoo! Groups (domaindrivendesign)*. Consultado em 15 de Julho de 2010
3. Larman 2005 chapter 17, section 11.
4. Larman 2005, pp. 314–315.

Referências

- Larman, Craig (2005) [2004]. *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development* (http://authors.phptr.com/larman/uml_ooad/index.html) 3rd ed. New Jersey: Prentice Hall. ISBN 0-13-148906-2