Design Paterns

Design Paterns

- Aplicações empresariais são complexas, isto é um fato.

Diariamente desenvolvedores, arquitetos, gerentes de projeto e usuários são forçados a lidar com estruturas de dados complexas, alterações em regras de negócio, mudanças de necessidades dos usuários e novas tecnologias.

Naturalmente, os desenvolvedores geralmente têm menos tempo e menos recursos do que precisariam (ou gostariam) para enfrentar tudo isso.

Design Paterns

- Existem várias formas possíveis de tratar tal complexidade, mas todas podem ser agrupadas em dois princípios: utilizar abordagens comprovadamente funcionais e antecipar necessidades futuras. Em ambos os casos, existem técnicas comuns que transcende um sistema individual.
- Essas técnicas são conhecidas como design patterns.
- Esses patterns podem ser usados repetidas vezes, facilitando a incorporação da experiência de desenvol-vimentos anteriores em novos projetos.

Design Paterns

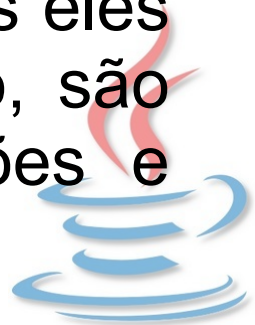
- Patterns também fornecem uma linguagem comum para a discussão de arquitetura de software em um nível mais elevado.
- Apenas dizer que um sistema implementa o pattern factory, por exemplo, pode comunicar uma decisão de projeto muito mais rapidamente e mais precisamente do que uma explicação complexa, assumindo que ambos os lados saibam o que é o padrão factory.
- Com a utilização cada vez maior da linguagem Java e os design patterns têm se tornado mais populares, uma vez que para construir uma aplicação de grande porte é imprescindível a utilização de patterns adequados.

Camadas da aplicação

- Aplicações empresariais modernas geralmente mantêm no mínimo três camadas: um cliente, um servidor middleware e um banco de dados.
- Projetos em multi-camadas resolvem vários problemas, mas também podem criar alguns outros.
- Separar interface do usuário, lógica de negócio e dados da aplicação torna o projeto como um todo mais fácil de ser compreendido e permite que diferentes equipes trabalhem em paralelo em cima de diferentes componentes, mas há um lado ruim: aumenta o overhead de comunicação entre as camadas.

Camadas da aplicação

- Projetos em multi-camadas possuem mais componentes para serem monitorados e o sucesso de muitos projetos em multi-camadas se enforcam ao tentar dividir as tarefas em componentes apropriados e ao organizar apropria-damente os resultados.
- Os patterns ajudarão você a melhorar a performance de seus sistemas em multi-camadas e torná-los mais fáceis de se manter ao reduzir a complexidade. Como todos eles estão relacionados ao particionamento da aplicação, são patterns essenciais para quase todas as aplicações e altamente relevantes para todos os desenvolvedores.



MVC(Model-View-Controller)

- O pattern **Model-View-Controller (MVC)** reforça um projeto modular e de fácil manutenção e força a separação de camadas.
- O design pattern MVC permite que você separe o modelo de dados das várias formas que o dado pode ser acessado e manipulado. Um sistema MVC é dividido em um modelo de dados, um conjunto de visões e um conjunto de controladores.
- As visões fornecem a interface do usuário. O controlador é geralmente implementado como uma ou mais classes Java. O modelo consiste de um código que fornece acesso direto ao banco de dados relacional.

MVC(Model-View-Controller)

- MVC é útil principalmente para aplicações grandes e distribuídas onde dados idênticos são visualizados e manipulados de formas variadas.
- Como o MVC facilita a divisão de trabalho por conjuntos de habilidades, este pattern é bastante adequado para empresas de desenvolvimento que suportam desenvol-vimento modular e concorrente com muitos desenvol-vedores.
- Promovendo a portabilidade de interfaces e do back-end, o pattern MVC também torna fácil testar e manter suas aplicações.

MVC(Model-View-Controller)

- A chave para o MVC é a separação de responsabilidades.
- As visões podem usar o modelo de dados para exibir resultados, mas elas não são responsáveis por atualizar o banco de dados.
- Os controladores são responsáveis por selecionar uma visão apropriada e por fazer alterações no modelo de dados.
- O modelo, por sua vez, é responsável por representar os dados base da aplicação.

MVC(Model-View-Controller)

- Algumas vezes o modelo de dados também incluirá a lógica de negócio (as regras para manipular os dados de negócio), e algumas vezes a lógica de negócio existirá na camada do controlador.
- Ao utilizar um padrão de projeto queremos alcançar objetivos da engenharia de software usando classes e métodos em linguagens orientadas a objeto.
- Padrão é a maneira testada ou documentada de alcançar um objetivo qualquer.