Monitoria Desenho

Padrões de projetos GoFs

Agenda

- Observer
- Template
- State
- Strategy

Próxima Monitoria

- Decorator
- Adapter
- Proxy
- Facade

Observer

Intenção: Definir uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados.

Aplicabilidade: Use o padrão observer em qualquer uma das seguintes situações

- Quando uma mudança em um objeto exige mudanças em outros, e você não sabe quantos objetos necessitam ser mudados
- Quando um objeto deveria ser capaz de notificar outros objetos sem faer hipóteses, ou usar informações, sobre quem são esses objetos. Em outras palavras, você não quer que esses objetos sejam fortemente acoplados

Diagrama de classe da literatura

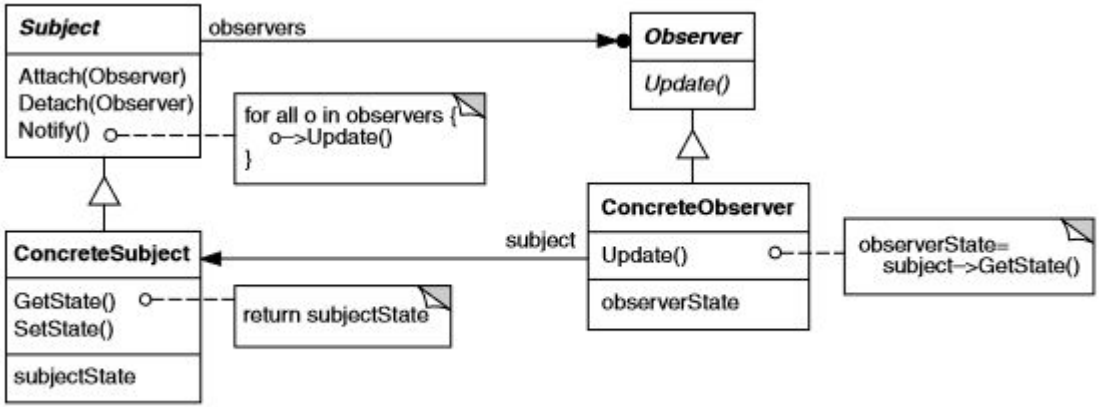


Diagrama de classe da literatura

• Subject – conhece os seus observadores. Um número qualquer de objetos Observer pode observar um subject. – fornece uma interface para acrescentar e remover objetos, permitindo associar e desassociar objetos observer.

• Observer – define uma interface de atualização para objetos que deveriam ser notificados sobre mudanças em um Subject.

• ConcreteSubject – armazena estados de interesse para objetos ConcreteObserver. – envia uma notificação para os seus observadores quando seu estado muda.

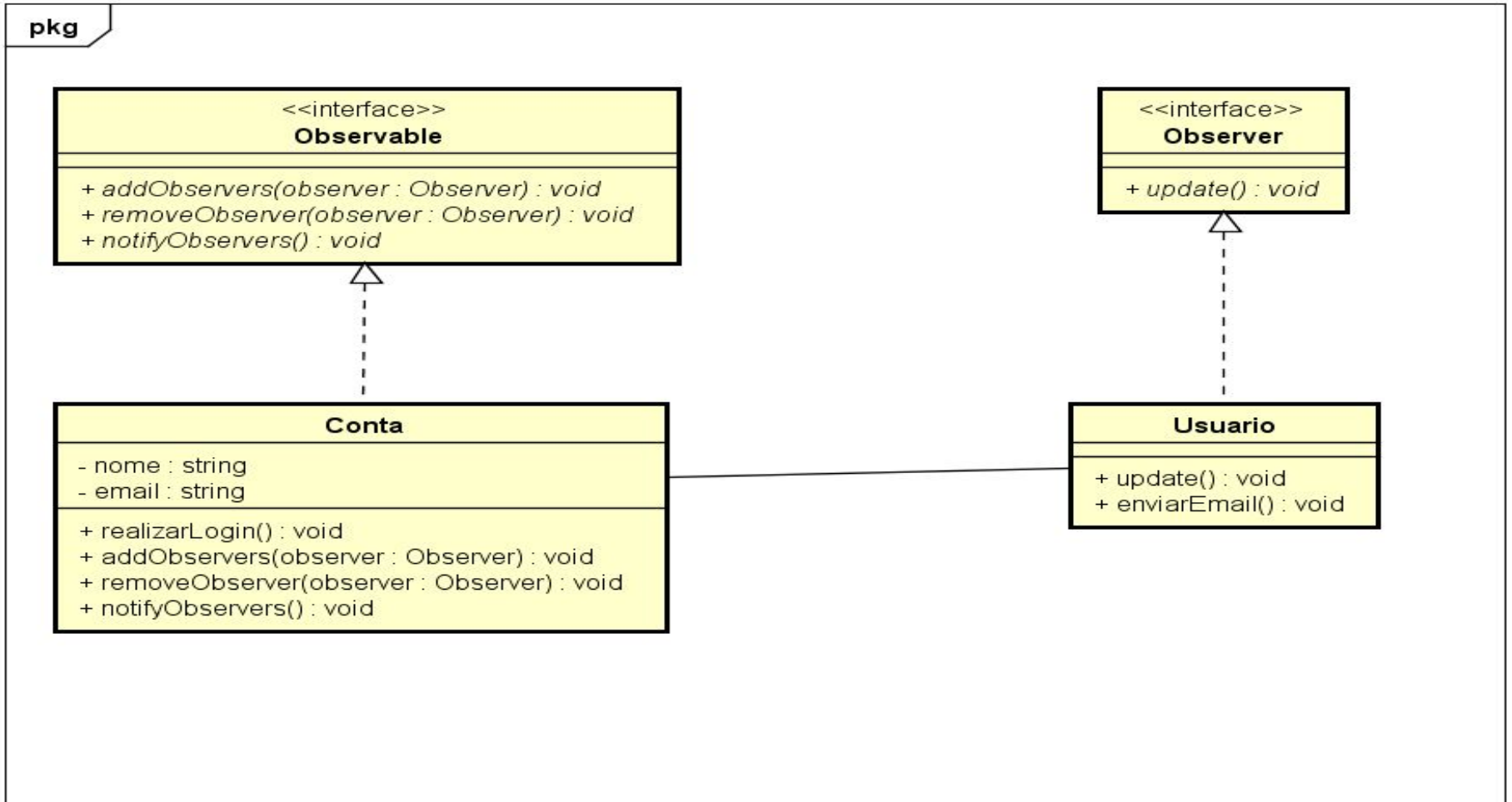
• ConcreteObserver – mantém uma referência para um objeto ConcreteSubject. – armazena estados que deveriam permanecer consistentes com os do Subject. – implementa a interface de atualização de Observer, para manter seu estado consistente com o do subject.

Questão 5 da lista

Você foi contratado pela Blizzard para desenvolver um jogo. Num determinado momento, foi lhe passado a seguinte história de usuário: Eu como jogador, desejo ser notificado por email sempre que minha conta for acessada, para minha segurança. Proponha um padrão de projeto que melhor solucione tal história e:

a. Desenhe o Diagrama de Classes

b. Faça um pseudo-código em Java para tal problema


pkg
<<interface>>
Observable
+ addObservers(observer : Observer) : void
+ removeObserver(observer : Observer) : void
+ notifyObservers() : void
<<interface>>
Observer
+ update() : void
Conta
- nome : string
- email : string
+ realizarLogin() : void
+ addObservers(observer : Observer) : void
+ removeObserver(observer : Observer) : void
+ notifyObservers() : void
Usuario
+ update() : void
+ enviarEmail() : void

Template Method

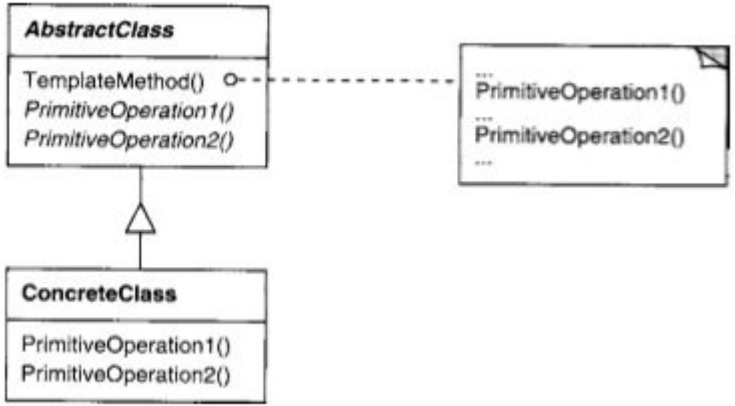
Intenção:Definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. Template Method permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo.

O padrão Template Method pode ser usado:

• para implementar as partes invariantes de um algoritmo uma só vez e deixar para as subclasses a implementação do comportamento que pode variar.

• para controlar extensões de subclasses. Você pode definir um método-template que chama operações “gancho” (ver Conseqüências) em pontos específicos, desta forma permitindo extensões somente nesses pontos.

Diagrama de Classe



Participantes

• AbstractClass (Application)

– define operações primitivas abstratas que as subclasses concretas definem para implementar passos de um algoritmo. – implementa um método-template que define o esqueleto de um algoritmo. O método-template invoca operações primitivas, bem como operações definidas em AbstractClass ou ainda outros objetos.

• ConcreteClass (MyApplication) – implementa as operações primitivas para executarem os passos específicos do algoritmo da subclasse.

Strategy

Intenção: Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam.

Use o padrão Strategy quando:

• muitas classes relacionadas diferem somente no seu comportamento. As estratégias fornecem uma maneira de configurar uma classe com um dentre muitos comportamentos;

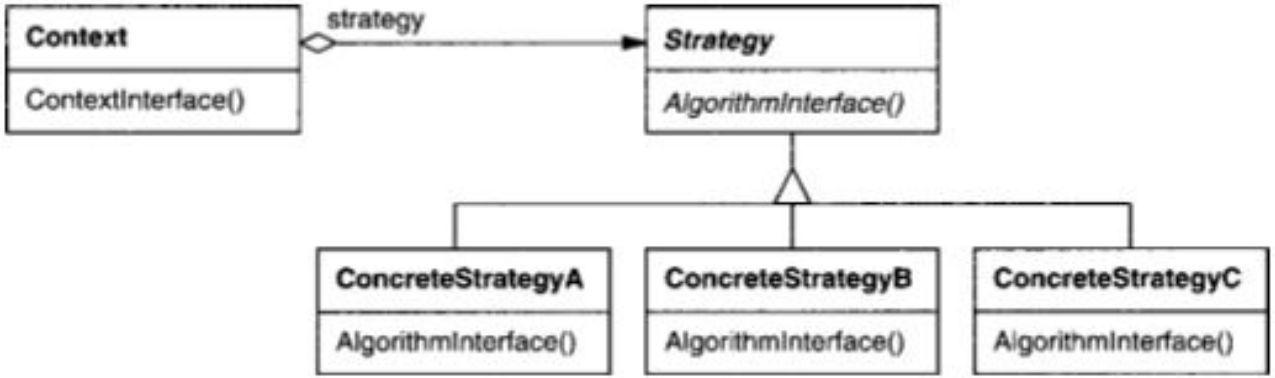
• você necessita de variantes de um algoritmo. Por exemplo, pode definir algoritmos que refletem diferentes soluções de compromisso entre espaço/ tempo. As estratégias podem ser usadas quando essas variantes são implementadas como uma hierarquia de classes de algoritmos [HO87];

Strategy

• um algoritmo usa dados dos quais os clientes não deveriam ter conhecimento. Use o padrão Strategy para evitar a exposição das estruturas de dados complexas, específicas do algoritmo;

• uma classe define muitos comportamentos, e estes aparecem em suas operações como múltiplos comandos condicionais da linguagem. Em vez de usar muitos comandos condicionais, mova os ramos condicionais relacionados para a sua própria classe Strategy.

Diagrama de classes



Participantes

• Strategy – define uma interface comum para todos os algoritmos suportados. Context usa esta interface para chamar o algoritmo definido por uma ConcreteStrategy.

• ConcreteStrategy – implementa o algoritmo usando a interface de Strategy.

• Context – é configurado com um objeto ConcreteStrategy;

– mantém uma referência para um objeto Strategy;

– pode definir uma interface que permite a Strategy acessar seus dados.

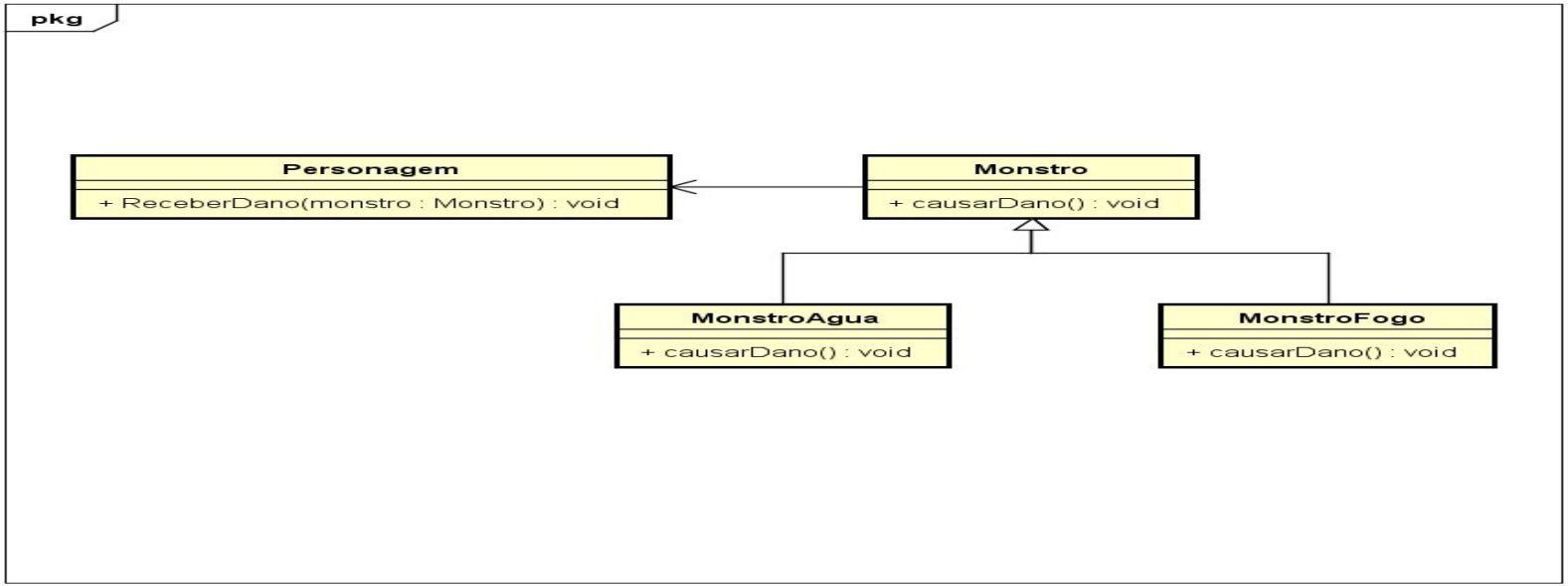
Questão 6 da Lista

Ainda no desenvolvimento do jogo, o engenheiro chefe da Blizzard lhe pediu para solucionar o seguinte problema: O personagem precisa receber 3x mais dano dos monstros de água e 1/2x mais dano dos monstros do tipo fogo. Proponha um padrão de projeto que melhor resolva este problema e:

a. Desenhe o Diagrama de Classes

b. Faça um pseudo-código em Java para tal problema

Diagrama de classe



State

Intenção

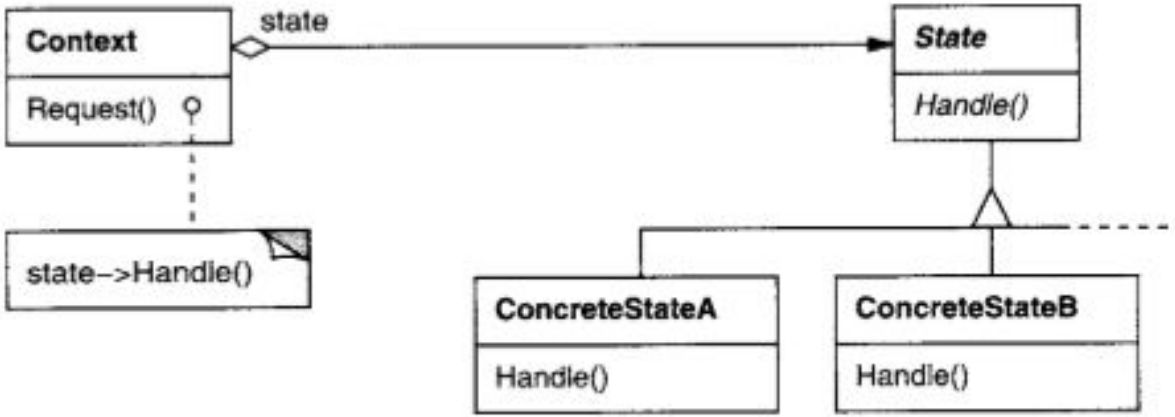
Permite a um objeto alterar seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado sua classe.

Use o padrão State em um dos dois casos seguintes:

• o comportamento de um objeto depende do seu estado e ele pode mudar seu comportamento em tempo de execução, dependendo desse estado;

• operações têm comandos condicionais grandes, de várias alternativas, que dependem do estado do objeto. Esse estado é normalmente representado por uma ou mais constantes enumeradas. Freqüentemente, várias operações conterão essa mesma estrutura condicional. O padrão State coloca cada ramo do comando adicional em uma classe separada. Isto lhe permite tratar o estado do objeto como um objeto propriamente dito, que pode variar independentemente de outros objetos.

Diagrama de classe



Obrigado