BSI FACOM UFU

Programação Orientada a Objetos 2

Lista de Exercícios 2

Padrões de Projeto: Decorator, Proxy, Fachada, Observer, Template, Strategy

Injeção de Dependências

Frameworks

Profa. Fabíola S. F. Pereira

QUESTÃO 1 Para cada assertiva a seguir, assinale verdadeiro ou falso. **Quando falso, justifique.**

a. ( ) Proxy é um padrão que funciona como um intermediário que controla o acesso a um objeto base.
b. ( ) Um proxy tem que encaminhar todas as chamadas realizadas em métodos de sua interface para métodos equivalentes do objeto base, podendo, no entanto, realizar algum processamento antes ou depois desse encaminhamento.
c. ( ) Strategy é um padrão que permite converter a interface de uma classe para outra interface esperada pelos seus clientes. Ele viabiliza então que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.
d. ( ) Uma desvantagem de Fachadas é aumentar o acoplamento entre um subsistema e seus clientes.
e. ( ) Decorador é um padrão que permite converter a interface de uma classe para outra interface esperada pelos seus clientes. Ele viabiliza então que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.
f. ( ) Um adaptador oferece uma interface diferente para o objeto adaptado. Em contraste, um proxy possui a mesma interface que o seu objeto base.
g. ( ) Como são padrões distintos, uma Fachada não pode ser implementada como um Singleton.

QUESTÃO 2 Dado o código abaixo de uma classe Subject (do padrão Observador):

```
interface Observer {
  public void update(Subject s);
}

class Subject {

  private List<Observer> observers=new ArrayList<Observer>();

  public void addObserver(Observer observer) {
```

```
    observers.add(observer);
  }

  public void notifyObservers() {
    (A)
  }

}
```

Implemente o código de notifyObservers, comentado com um (A) acima.

QUESTÃO 3 Suponha uma classe base A. Suponha que queremos adicionar quatro funcionalidades opcionais F1, F2, F3 e F4 em A. Essas funcionalidades podem ser adicionadas em qualquer ordem, isto é, a ordem não é importante. Se usarmos herança, quantas subclasses de A teremos que implementar? Se optarmos por uma solução por meio de decoradores, quantas classes teremos que implementar (sem contar a classe A). Justifique e explique sua resposta.

QUESTÃO 4 Suponha uma classe Documento cujos objetos podem ser salvos em disco de forma compactada. Atualmente, usamos um algoritmo de compactação X. Mas amanhã existe uma chance de que precisaremos substituir completamente X por um novo algoritmo. Qual padrão de projeto você recomenda usar neste caso? Justifique.

QUESTÃO 5 Suponha que temos uma classe Stock, que representa ações de uma empresa negociada em bolsas de valores. Diversos clientes (bancos, corretoras, clientes pessoa física, etc) querem ser notificados toda vez que o preço de uma ação sofrer uma alteração significativa. Qual padrão de projeto você recomenda usar neste caso? Justifique.

QUESTÃO 6 Injeção de Dependência, muitas vezes, é comparada com padrão de projeto Fábrica. Qual a desvantagem de configurar dependências por meio de fábricas? Para responder, compare os seguintes códigos:

```
class A {
  IB b;
  A(IB b) {
    this.b = b; // injeção de dependência
  }
}

class A {
  IB b;
  A() {
    this.b = IB_Factory.getInstance(); // fábrica
  }
}
```

QUESTÃO 7 Qual a relação entre Injeção de Dependência (padrão de projeto) e Inversão de Dependência (princípio de projeto)?

QUESTÃO 8 Dê o nome dos seguintes padrões de projeto e diga se trata-se de um padrão criacional, comportamental ou estrutural.

1. Permite parametrizar os algoritmos usados por uma classe.
2. Torna uma estrutura de dados aberta a extensões, isto é, permite adicionar uma função em cada elemento de uma estrutura de dados, mas sem alterar o código de tais elementos:
3. Permite que um objeto avise outros objetos de que seu estado mudou:
4. Define o esqueleto de um algoritmo em uma classe base e delega a implementação de alguns passos para subclasses:
5. Oferece uma interface unificada e de alto nível que torna mais fácil o uso de um sistema.
6. Permite adicionar dinamicamente novas funcionalidades a uma classe.
7. Funciona como um intermediário que controla o acesso a um objeto base