

Resolução de Exercícios

Lista 2

Profa. Fabíola S. F. Pereira

fabiola.pereira@ufu.br

LISTA 2

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)  
    }  
}
```

³Implemente o código de notifyObservers, comentado com um (A) acima.

QUESTÃO 2

```
public void notifyObservers() {  
    Iterator it = observers.iterator();  
    while (it.hasNext()) {  
        Observer obs= (Observer) it.next();  
        obs.update(this);  
    }  
}
```

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 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 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.

Strategy

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 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 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.

Observer

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  
    }  
}
```

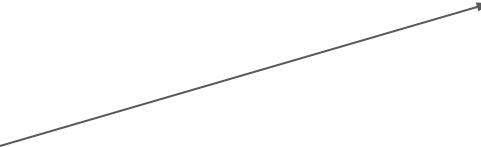
QUESTÃO 6

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  
    }  
}
```

Como seria o método
getInstance()?




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  
    }  
}
```

E se quiséssemos outros tipos
B1, B2?



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  
    }  
}
```

Pode-se injetar objetos de qualquer tipo que implemente a interface IB. Ou seja, não precisa ser apenas objetos de um tipo B.

Logo, alguns objetos do tipo A podem depender de B1, outros de B2, outros ainda de B3, etc. Assumindo que B1, B2 e B3 são classes que implementam a interface IB

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

Vai sempre retornar um objeto de um determinado tipo, digamos B.

Ou seja, todo objeto do tipo A vai ter uma dependência para um objeto do tipo B

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. Converte a interface de uma classe para outra interface esperada pelos clientes. Permite que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.
7. Funciona como um intermediário que controla o acesso a um objeto base

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. **Strategy, comportamental**
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. **Visitor, comportamental**
3. Permite que um objeto avise outros objetos de que seu estado mudou. **Observer, comportamental**
4. Define o esqueleto de um algoritmo em uma classe base e delega a implementação de alguns passos para subclasses. **Template, comportamental**
5. Oferece uma interface unificada e de alto nível que torna mais fácil o uso de um sistema. **Fachada, estrutural**
6. Funciona como um intermediário que controla o acesso a um objeto base **Proxy, estrutural**

FIM