



### Quizz 3

INFORMAÇÕES DOCENTE						
CURSO: ENGENHARIA DE SOFTWARE	DISCIPLINA: PROJETO DE SOFTWARE	TURNO	MANHÃ <input checked="" type="checkbox"/>	TARDE <input type="checkbox"/>	NOITE <input checked="" type="checkbox"/>	PERÍODO/SALA: 4º
<b>PROFESSOR (A):</b> João Paulo Carneiro Aramuni						

### Padrões GRASP – General Responsibility Assignment Software Patterns

#### Especialista

1) Considere um sistema de e-commerce em que é necessário implementar o método `calcularTotalCompra()`, que deve somar o valor total de todos os itens comprados (considerando a quantidade e o preço de cada item). Com base no padrão GRASP de Especialista, qual classe é a mais apropriada para armazenar esse método?

```
class Cliente {  
    List<Compra> compras;  
}
```

```
class Compra {  
    List<Item> itens;  
}
```

```
class Item {  
    double preco;  
    int quantidade;  
}
```

- A) Cliente, pois ele contém uma lista de compras associadas.
- B) Compra, pois ela possui os itens que compõem a compra.
- C) Item, pois ele armazena o preço e a quantidade.
- D) Nenhuma das classes, pois seria melhor criar uma classe separada para essa operação.
- E) Todas as classes, pois cada uma tem parte dos dados necessários.



**PUC Minas**

2) Dado o código a seguir, qual classe deveria implementar o método calcularMedia() para ser considerada Especialista?

```
class Aluno {  
    List<Nota> notas;  
}  
  
class Nota {  
    double valor;  
}  
  
class Professor {  
    String nome;  
}
```

A) Aluno  
B) Nota  
C) Professor  
D) Nenhuma das classes  
E) Todas as classes

### Criador

3) Em um sistema de vendas, é necessário criar novas instâncias de Pedido sempre que um cliente realiza uma compra. Considerando o padrão GRASP de Criador, qual classe seria a mais apropriada para criar instâncias de Pedido?

```
class Cliente {  
    String nome;  
    List<Pedido> pedidos;  
}  
  
class Pedido {  
    List<Item> itens;  
}  
  
class Item {  
    String descricao;  
    double preco;  
}
```



## PUC Minas

- A) Cliente, pois ele possui uma lista de pedidos associados ao cliente.
  - B) Pedido, pois ele representa diretamente a compra realizada.
  - C) Item, pois ele contém detalhes dos produtos comprados.
  - D) Todas as classes, pois cada uma possui dados relevantes para a criação do pedido.
  - E) Nenhuma das classes.
- 4) Em um sistema bancário, sempre que uma transação é realizada, uma nova instância de Transacao deve ser criada. Considerando o padrão GRASP de Criador, qual classe seria a mais apropriada para ser responsável pela criação de uma instância de Transacao?

```
class Conta {  
    double saldo;  
    List<Transacao> transacoes;  
}
```

```
class Transacao {  
    double valor;  
    Date data;  
}
```

```
class Banco {  
    String nome;  
    List<Conta> contas;  
}
```

- A) Conta, pois ela gerencia as transações associadas ao seu saldo.
- B) Transacao, pois ela representa a própria transação.
- C) Banco, pois ele possui uma lista de contas e gerencia as transações do sistema.
- D) Nenhuma das classes.
- E) Todas as classes.

### Coesão alta

- 5) Qual prática deve ser evitada para garantir coesão alta em uma classe?
- A) Dividir uma classe grande em classes menores com responsabilidades específicas.
  - B) Atribuir uma única responsabilidade à classe.
  - C) Colocar múltiplos métodos sem relação entre si em uma mesma classe.
  - D) Garantir que a classe tenha responsabilidade focada.
  - E) Manter apenas atributos relacionados diretamente à responsabilidade da classe.



## PUC Minas

6) Qual das opções abaixo pode reduzir a coesão de uma classe?

- A) Remover métodos que não estão relacionados à responsabilidade principal.
- B) Adicionar métodos para diferentes funcionalidades não relacionadas.
- C) Manter o foco da classe em uma única responsabilidade.
- D) Dividir a classe em classes menores e mais específicas.
- E) Evitar adicionar métodos e atributos desnecessários.

### Acoplamento fraco

7) Dado o código a seguir, qual modificação ajudaria a reduzir o acoplamento entre Banco e Conta?

```
class Banco {  
    Conta conta;  
}
```

```
class Conta {  
    double saldo;  
}
```

- A) Fazer Conta herdar de Banco.
- B) Adicionar um método `getSaldo()` na classe Banco.
- C) Usar uma interface para Conta.
- D) Adicionar o saldo diretamente na classe Banco.
- E) Tornar conta um atributo estático.

8) Qual técnica reduz o acoplamento entre classes em um sistema?

- A) Herança entre classes.
- B) Dependência direta em classes concretas.
- C) Uso de interfaces.
- D) Definição de muitos métodos públicos.
- E) Armazenar dados em classes não relacionadas.



## PUC Minas

### Controlador

9) Em uma aplicação de e-commerce, deseja-se implementar o caso de uso "finalizar pedido". Esse processo inclui:

- Verificar o status do pagamento do pedido.
- Atualizar o status do pedido para "Finalizado".
- Enviar uma notificação ao cliente com a confirmação do pedido.

Dado o código a seguir, qual classe seria a melhor escolha para centralizar a lógica de controle desse processo, de acordo com o padrão GRASP de Controlador?

```
class GestorDePedidos {  
    Pedido pedido;  
    Cliente cliente;  
}  
  
class Pedido {  
    void atualizarStatus(String status) /* Implementação */;  
    boolean verificarPagamento() /* Implementação */;  
}  
  
class Cliente {  
    String email;  
    void enviarNotificacao(String mensagem) /* Implementação */;  
}
```

- A) Pedido, pois ele tem o método para atualizar o status e pode verificar o pagamento.
- B) GestorDePedidos, pois ele pode coordenar todas as ações necessárias para finalizar o pedido.
- C) Cliente, pois ele deve ser notificado sobre o pedido finalizado e pode enviar uma resposta.
- D) Nenhuma das classes, pois seria melhor ter uma classe separada para gerenciar o processo de finalização.
- E) Todas as classes, pois cada uma deveria implementar uma parte do processo.



## PUC Minas

10) Em um sistema bancário, é necessário implementar o caso de uso "realizar transação", que deve:

- Verificar se a conta de origem possui saldo suficiente.
- Debitar o valor da conta de origem.
- Creditar o valor na conta de destino.

Considerando as classes abaixo, qual seria a melhor escolha para centralizar a lógica de controle desse caso de uso, segundo o padrão GRASP de Controlador?

```
class Banco {  
    List<Conta> contas;  
  
    void processarTransacoes() { /* Implementação */ }  
}  
  
class Conta {  
    double saldo;  
  
    void debitar(double valor) { /* Implementação */ }  
    void creditar(double valor) { /* Implementação */ }  
}  
  
class Transacao {  
    Conta contaOrigem;  
    Conta contaDestino;  
    double valor;  
  
    boolean validar() { /* Verifica saldo e limites */ }  
    void executar() { /* Implementação */ }  
}
```

- A) Banco, pois ele possui uma lista de contas e pode manipular os saldos.
- B) Conta, pois ela deve gerenciar suas próprias transações e saldo.
- C) Transacao, pois ela representa o processo de transferência e tem acesso às contas envolvidas.
- D) Nenhuma das classes, pois seria necessário criar uma nova classe para controlar a transação.
- E) Todas as classes, pois cada uma deveria gerenciar uma parte do processo.



## PUC Minas

Gabarito:

- 1) B
- 2) A
- 3) A
- 4) A
- 5) C
- 6) B
- 7) C
- 8) C
- 9) B
- 10) C

### Questões de Verdadeiro ou Falso sobre Padrões GRASP

#### Especialista

- 1) No padrão GRASP de Especialista, a responsabilidade de calcular um valor total deve ser atribuída à classe que possui os dados necessários para realizar esse cálculo.

V ou F: \_\_\_\_\_

- 2) A aplicação do padrão Especialista pode levar ao aumento de acoplamento entre as classes, já que ele incentiva uma classe a acessar dados diretamente em outras classes.

V ou F: \_\_\_\_\_

#### Criador

- 3) Segundo o padrão Criador, a classe 'Pedido' deve ser responsável por criar instâncias de 'Item', pois cada 'Pedido' pode ter múltiplos itens associados a ele.

V ou F: \_\_\_\_\_

- 4) O padrão Criador sugere que uma classe deve ser responsável por criar outra classe se ela utiliza diretamente a instância da classe a ser criada, tem uma relação de composição ou agregação com ela, ou possui os dados necessários para a criação.

V ou F: \_\_\_\_\_



## PUC Minas

### Coesão Alta

5) No GRASP, uma alta coesão indica que as responsabilidades de uma classe estão intimamente relacionadas, o que ajuda a tornar o código mais fácil de entender e de manter.

V ou F: \_\_\_\_\_

6) A coesão alta sempre leva a um design onde as classes têm muitas responsabilidades, pois isso ajuda a reduzir a quantidade de classes no sistema.

V ou F: \_\_\_\_\_

### Acoplamento Fraco

7) O padrão de Acoplamento Fraco visa reduzir as dependências entre classes, tornando-as mais independentes e facilitando a manutenção do código.

V ou F: \_\_\_\_\_

8) No GRASP, um acoplamento fraco significa que uma classe deve depender fortemente de várias outras para realizar suas tarefas, garantindo que todas as funcionalidades necessárias estejam em uma única classe.

V ou F: \_\_\_\_\_

### Controlador

9) De acordo com o padrão Controlador, é apropriado que a classe 'PedidoController' seja responsável por orquestrar as operações para finalizar um pedido em um sistema de e-commerce.

V ou F: \_\_\_\_\_

10) O padrão Controlador sugere que um controlador específico deve acessar diretamente todos os detalhes das entidades do sistema para realizar uma operação.

V ou F: \_\_\_\_\_

### Gabarito explicado

1) \*\*V\*\* - O padrão Especialista atribui a responsabilidade de uma tarefa à classe que possui a maior parte dos dados necessários para realizá-la, tornando o código mais claro e fácil de manter.



## PUC Minas

- 2) **\*\*F\*\*** - Embora o Especialista possa acessar dados de outras classes, ele não deve gerar um aumento de acoplamento indesejado. Em vez disso, ele centraliza a responsabilidade em uma classe que já possui os dados, reduzindo a necessidade de interação com várias classes.
- 3) **\*\*V\*\*** - Pelo padrão Criador, uma classe deve ser responsável por criar instâncias de objetos que ela contém ou utiliza fortemente. Como a classe Pedido agrupa múltiplos Itens, faz sentido que ela seja responsável por criar esses objetos, mantendo o baixo acoplamento e alta coesão no sistema.
- 4) **\*\*V\*\*** - O padrão Criador recomenda que uma classe seja responsável por criar outra se ela tem uma relação de composição/agregação, usa diretamente, ou possui os dados necessários para inicializar a instância.
- 5) **\*\*V\*\*** - Alta coesão significa que a classe tem responsabilidades intimamente relacionadas, o que ajuda na legibilidade, manutenção e reutilização do código, facilitando o entendimento do que a classe faz.
- 6) **\*\*F\*\*** - Alta coesão não implica que uma classe deve ter muitas responsabilidades. Pelo contrário, a coesão alta sugere que as responsabilidades de uma classe sejam focadas em um único propósito ou conjunto de tarefas relacionadas.
- 7) **\*\*V\*\*** - Acoplamento fraco é sobre reduzir dependências, o que facilita a manutenção e permite que as classes sejam usadas independentemente em diferentes contextos.
- 8) **\*\*F\*\*** - Acoplamento fraco indica que uma classe deve evitar dependências excessivas em outras classes. Depender de várias classes diretamente para realizar tarefas não é acoplamento fraco, e sim acoplamento forte.
- 9) **\*\*V\*\*** - O padrão Controlador recomenda o uso de uma classe controladora (como 'PedidoController') para orquestrar operações de alto nível, como finalizar um pedido, pois ela atua como um intermediário entre a interface de usuário e a lógica de negócios.
- 10) **\*\*F\*\*** - Um controlador não deve acessar diretamente todos os detalhes de outras entidades; ele deve delegar tarefas para as entidades que são responsáveis por suas próprias operações, mantendo o princípio de coesão alta e acoplamento fraco.