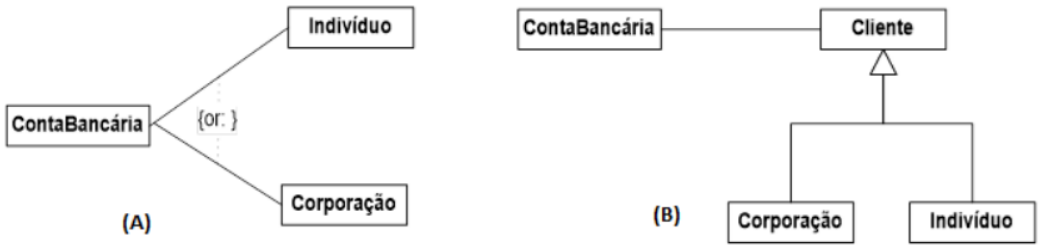
Lista de Exercícios I

Aluno: Rodolfo Oliveira Miranda

Questão 1 Quais são os padrões GRASP (ou padrões de responsabilidade geral)? Explique de maneira suscinta, cada um deles.

Questão 2 Considere os diagramas de classes de análise fornecidos na Figura 1, nos itens (a) e (b) abaixo, ambos de acordo com a notação da UML. Esses diagramas desejam representar o fato de que uma conta bancária pode estar associada a uma pessoa, que pode ser ou uma pessoa física (representada pela classe Indivíduo), ou uma pessoa jurídica (representada pala classe Corporação). Uma dessas duas soluções é melhor que a outra? Se sim, qual delas e em que sentido? Justifique sua resposta considerando alguns dos padrões GRASP.



Figuur 1: Diagramas com as duas soluções.

Questão 3 Qual a responsabilidade correta relacionada ao padrão GRASP Creator (Criador)?

Questão 1

GRASP é um conjunto de padrões que ajudam na atribuição de responsabilidades em objetos, visando design coeso,, baixo acoplamento e alta reutilização de software orientado a objetos.

- **Padrões Básicos**
  - **Information Expert**: Atribui responsabilidade ao objeto que possui as informações necessárias para realizar a tarefa
  - **Creator**: Define quem deve ser responsável por criar uma instância de uma classe. Geralmente, é a classe que usa ou agrega o objeto a ser criado.
  - **High Cohesion**: Garante que as responsabilidades dentro de uma classe sejam bem relacionadas, evitando que ela se torne muito complexa ou sobrecarregada.
  - **Low Coupling**: Promove a redução de dependências entre classes, facilitando a manutenção e a extensibilidade do sistema.

    - **Controller**: Determina quem deve lidar com eventos do sistema. Geralmente, um objeto intermediário que dele tarefas para outras classes.

- **Padrões Avançados**
    - **Polymorphism**: Usa herança e interfaces para definir comportamentos baseados em tipos dinâmicos, favorecendo extensibilidade.
    - **Pure Fabrication**: Criação de classes artificiais que não pertencem diretamente ao domínio, mas que melhorar a estrutura do design.
    - **Indirection**: Introduz um intermediário para desacoplar classes, promovendo flexibilidade e facilidade de manutenção.
    - **Protected Variations:** Protege o sistema contra mudanças em partes específicas usando abstrações e encapsulamento.

Questão 2

A segunda representação é a melhor maneira de se desenvolver este sistema, pois com uma interface “Cliente” diminuímos o acoplamento da classe “” (Low Coupling) e criamos a possibilidade de, futuramente, derivarmos outras classes que implementem a classe “Cliente” (Polymorphism). Além de especificarmos uma classe onde iremos instanciar suas derivantes (Creator).

Questão 3

Como dito na questão um, o padrão Creator é responsável por determinar qual classe será responsável por criar instâncias de outra classe. A decisão é baseada na relação entre as classes e segue o princípio de design que facilita o entendimento e a manutenção do código.

Segundo ele, uma classe A deve ser responsável de por criar uma instância de outra classe B se um ou mais das seguintes condições forem verdadeiras.

1. **Agregação**: A classe A agrega ou contém instâncias de B.
    - Exemplo: Um objeto Pedido pode criar instâncias de ItemPedido porque agrega esses objetos.
2. **Associação**: A classe A utiliza ou é associada à classe B.
    - Exemplo: Um objeto Cliente pode criar uma instância de Contrato porque ele está associado ao contrato.

3. **Inicialização**: A classe A precisa fornecer dados ou parâmetros para inicializar objetos da classe B.
    - Exemplo: Um objeto Sessão pode criar uma instância de UsuárioLogado, fornecendo as informações necessárias.
4. **Dependência**: A classe A usa a classe B intensivamente para realizar suas responsabilidades.
    - Exemplo: Um objeto GerenciadorDeRelatórios pode criar instâncias de Relatório porque depende diretamente delas.
5. **Encapsulamento**: A classe A deseja ocultar o processo de criação de objetos B para promover flexibilidade e evitar acoplamento externo.

Ao implementar **Creator**, a ideia é manter o design coeso e promover baixo acoplamento. Isso facilita a manutenção e as futuras alterações no código.