Resolução do Exame de PDS

Parte 1: Conceitos Gerais (True/False & Perguntas Abertas)

Secção I: Verdadeiro ou Falso (True/False)

Nota: Algumas respostas marcadas na folha original parecem incorretas (perguntas 3 e 4). As respostas abaixo são as tecnicamente corretas.

1. **T (Verdadeiro)**: Em design de software, aumentar a complexidade leva sempre à diminuição da manutenção e legibilidade.
2. **T (Verdadeiro)**: A abstração ajuda a gerir a complexidade escondendo detalhes internos e expondo apenas interfaces necessárias.
3. **T (Verdadeiro)**: O uso de padrões de design ajuda a mitigar a complexidade fornecendo soluções padronizadas. *(Na folha original está marcado F, o que é incorreto).*
4. **T (Verdadeiro)**: Refactoring é o processo de reestruturar código existente sem alterar o seu comportamento externo. *(Na folha original está marcado F, o que é incorreto).*
5. **F (Falso)**: Gerir a complexidade não é apenas um desafio técnico; requer consideração de dinâmicas de equipa e gestão.
6. **T (Verdadeiro)**: O processo de design inclui requisitos, análise das necessidades do utilizador e restrições técnicas.
7. **F (Falso)**: As características de design interno focam-se na estrutura do código, não em funcionalidades externas ou UI.
8. **T (Verdadeiro)**: Os princípios GRASP são diretrizes para atribuir responsabilidades a classes e objetos.

Secção II: Perguntas Abertas (Open Questions)

OQ1: GRASP (General Responsibility Assignment Software Patterns)

São princípios que guiam a atribuição de responsabilidades a classes e objetos.

- **Information Expert:** Atribuir a responsabilidade à classe que tem a informação necessária.
- **Creator:** Quem deve criar uma instância de A? (Geralmente quem contém ou agrega A).
- **Controller:** O objeto que serve de ponto de entrada para operações do sistema (camada entre UI e lógica).
- **Low Coupling:** Manter as dependências baixas.
- **High Cohesion:** Manter as responsabilidades da classe focadas.

OQ2: Padrões Criacionais (Creational)

Lidam com o mecanismo de criação de objetos, abstraindo o processo de instanciação. O seu propósito é tornar o sistema independente de como os objetos são criados.

- *Exemplos:* Singleton, Factory Method, Abstract Factory, Builder, Prototype.

OQ3: Padrões Estruturais (Structural)

Lidam com a composição de classes e objetos. O seu propósito é simplificar o design identificando formas simples de realizar relacionamentos entre entidades.

- *Exemplos:* Adapter, Bridge, Composite, Decorator, Facade, Proxy.

OQ4: Padrões Comportamentais (Behavioral)

Lidam com a comunicação entre objetos. O seu propósito é gerir algoritmos, relacionamentos e responsabilidades.

- *Exemplos:* Observer, Strategy, Command, Iterator, State, Template Method.

Parte 2: Identificação Prática (Código e Cenários)

Análise de Código

1. **Código `DataSource` / `CompressionProcessor`:**
    - Usa aninhamento de objetos (`new A(new B(new C))`) para adicionar funcionalidades.
    - **Resposta: e. Decorator**
2. **Código `DatabaseConnection`:**
    - Tem construtor privado e método `getInstance`.
    - **Resposta: c. Singleton**

Cenários (Desafios)

3. **Products and Boxes (Produtos e Caixas):**
    - Estrutura em árvore onde caixas contêm produtos ou outras caixas (tratar individual e composto uniformemente).
    - **Resposta: f. Composite**
4. **Weather Monitoring (Monitorização do Tempo):**
    - Mudanças num objeto são comunicadas a um conjunto de outros objetos.
    - **Resposta: h. Observer**

Parte 3: Escolha Múltipla - Criacionais e Estruturais

1. **Factory Method:** A) Allows dynamic creation of objects without specifying the exact class.
2. **Abstract Factory:** B) Create families of related objects without specifying their concrete classes.
3. **Builder:** B) To enable the creation of complex objects step by step.
4. **Prototype:** B) You need many instances of a class that share a common initial state (cloning).
5. **Singleton:** A) A class has only one instance and provides a global point of access.
6. **Adapter:** C) Converts one interface into another interface clients expect.
7. **Bridge:** B) Separate abstraction from its implementation so both vary independently.
8. **Composite:** B) Treat individual objects and compositions uniformly.
9. **Decorator:** B) Add responsibilities to objects dynamically.
10. **Facade:** B) Provide a unified interface to a set of interfaces in a subsystem.
11. **Flyweight:** B) Minimize memory usage by sharing data with similar objects.

Parte 4: Escolha Múltipla - Comportamentais e Estruturais

1. **Proxy:** B) Control access to an object by acting as a surrogate.
2. **Chain of Responsibility:** A) Multiple objects handle a request without sender knowing who handles it.
3. **Command:** B) Encapsulate a request as an object (parameterization/queuing).
4. **Iterator:** B) Access elements of an aggregate sequentially without exposing representation.
5. **Mediator:** C) Define an object that encapsulates how a set of objects interact.
6. **Memento:** B) Capture and externalize an object's internal state (restore later).
7. **Observer:** C) Allow objects to be notified of state changes in other objects.
8. **State:** A) An object alters its behavior when its internal state changes.
9. **Strategy:** B) Defines a family of algorithms, encapsulates each one, makes them interchangeable.
10. **Template Method:** A) Defines skeleton of algorithm, deferring steps to subclasses.
11. **Visitor:** C) Separate an algorithm from the object structure it operates on.