RESUMO PARA A PROVA FINAL DE ES 2

Conceitos de Projeto de Software

1. Histórico do Projeto de Software
    - A evolução das técnicas de desenvolvimento: da programação estruturada até os métodos ágeis.
    - Importância do projeto no ciclo de vida do software.
2. Metodologias
    - Tradicionais: Cascata, espiral, modelo incremental.
    - Ágeis: Scrum, XP (Extreme Programming), Kanban.
    - RAD (Rapid Application Development): Prototipação rápida para entregas mais velozes.

Projeto Orientado a Objetos (O.O.)

1. Fase Preliminar
    - Diagrama de Uso: Representação de atores e suas interações com os casos de uso.
    - Protótipos: Modelos iniciais para validar requisitos.
    - Diagrama de Robustez: Liga casos de uso a elementos do sistema (controladores, entidades e limites).
2. Fase Detalhada
    - Arquitetura Lógica (Domínio): Modelagem do negócio, incluindo classes e suas relações.
    - Arquitetura Física (Solução): Componentes físicos do sistema, como servidores e comunicação entre módulos.

Projeto de Interface/Usabilidade (Front-End)

1. Forma: Aspectos estéticos como cores, fontes e layout.
2. Estrutura: Navegação e organização lógica das telas.
3. Função (UX): Experiência e usabilidade voltadas à satisfação do usuário.

Projeto de Persistência (Back-End)

1. Mapeamento Objeto-Relacional (MOR): Conversão entre banco de dados relacional e objetos da aplicação.
    - Linguagens como OQL e JQL.
2. JPA (Java Persistence API): Frameworks como Hibernate para facilitar a persistência de objetos.

Arquitetura de Software

1. Uso de Frameworks
    - Caixa-Preta: O desenvolvedor utiliza as funcionalidades sem conhecer sua implementação interna.
    - Caixa-Branca: Permite a personalização e compreensão da lógica do framework.
2. Categorias
    - Front-End: Frameworks para interface (React, Angular).
    - Back-End: Frameworks para lógica de negócio e persistência (Spring, Django).

Critérios de Qualidade em Projeto de Software

1. Estimativas e Métricas
    - Pontos de Caso de Uso: Baseados em atores e complexidade dos casos.
        - Exemplo: Atores simples, médios e complexos possuem pesos diferentes.
    - Estimativas Ágeis: Priorizam flexibilidade e entregas rápidas (Scrum, Planning Poker).
2. Padrões de Projeto
    - Design Patterns GoF:
        - Criacionais: Factory, Builder, Singleton, Prototype.
        - Estruturais: Adapter, Bridge, Composite, Decorator.
        - Comportamentais: Observer, Strategy, State.
    - GRASP: Padrões que organizam responsabilidades (Controller, Expert, Baixo Acoplamento).
    - SOLID:
        - S: Single Responsibility Principle.
        - O: Open/Closed Principle.
        - L: Liskov Substitution Principle.
        - I: Interface Segregation Principle.
        - D: Dependency Inversion Principle.
3. Arquitetura Limpa
    - Separar preocupações usando camadas (Ex: Micro-Serviços).

Testes de Software

1. BDD (Behavior-Driven Development):
    - Baseado em cenários com a linguagem "gherkin" (Given, When, Then).
    - Realizado antes do desenvolvimento para garantir conformidade com requisitos.
2. CI/CD e DevOps
    - CI (Integração Contínua): Testes automatizados em cada alteração.
    - CD (Deploy Contínuo): Implementação automática de funcionalidades aprovadas.
    - DevOps: Integração entre desenvolvimento e operações para entrega contínua.

EXERCÍCIOS RESOLVIDOS

1. **Sistema de Agência de Empregos**

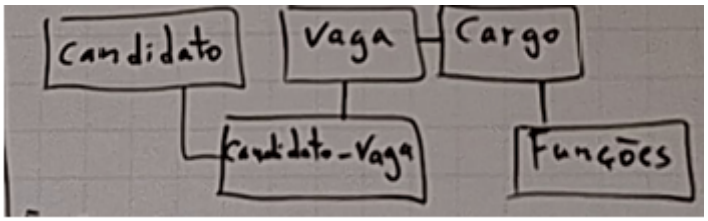
   Seja um sistema de agência de empregos com os seguintes casos de uso

   ○ cadastro de candidato
   ○ cadastro de empresas
   ○ cadastro de vagas da empresa
   ○ cadastro de cargos da empresa
   ○ cadastro de funções de um cargo
   ○ seleção de candidatos pré-aprovados para uma vaga

   ○ Sabendo-se que o uso de Seleção de Candidato é mais complexo e sabendo-se que é usada por 3 atores:

- gerente de RH
- gerente da área de candidato
- funcionário do RH

○ e que às classes usadas neste caso de uso são:



○ e que os indicadores são

- técnico = 0,19
- ambiente =0,87

○ temos valor de hora de desenvolvimento = R$ 200,00

A) Qual o custo do desenvolvimento ?

B) Sabendo que a empresa tem o histórico de 24h por Ponto de Caso de Uso, qual o prazo em dias (8h/dia) para o desenvolvimento ?

- Dados fornecidos:
  - Indicadores: Técnico = 0,19; Ambiente = 0,87.
  - Valor da hora: R$ 200,00.
  - Horas por Ponto: 24h.
- Passo 1: Determinar os Pontos de Caso de Uso (PCU).
  - Atores envolvidos:
    - Gerente de RH, gerente da área de candidato, funcionário do RH.
  - Complexidade do caso de uso: Seleção de candidatos pré-aprovados é complexo.
- Passo 2: Cálculo do custo.
  - Fórmula: Custo = PCU × Horas × Valor/h.
  - Resultado (exemplo com PCU = 3): Custo Total = 3 × 24 × 200 = R$ 14.400,00.
- Passo 3: Determinar o prazo em dias.
  - Fórmula: Prazo = Horas Totais / 8 (jornada de trabalho).
  - Resultado (exemplo com PCU = 3): Prazo = (3 × 24) / 8 = 9 dias.

---

2. Diferenças entre Padrões de Projeto

Qual a diferença entre os padrões:

○ Abstract Factory e Builder;
○ Singleton e Prototype;
○ Decoration e Observer;
○ Medidor e Bridge.

- Abstract Factory vs Builder:
  - Factory: Cria objetos relacionados.
  - Builder: Constroi objetos passo a passo.
- Singleton vs Prototype:
  - Singleton: Instância única.
  - Prototype: Cópia de objetos.
- Decorator vs Observer:
  - Decorator: Adiciona funcionalidades.
  - Observer: Notifica mudanças.
- Mediator vs Bridge:
  - Mediator: Coordena comunicação entre objetos.
  - Bridge: Separa abstração da implementação.

---

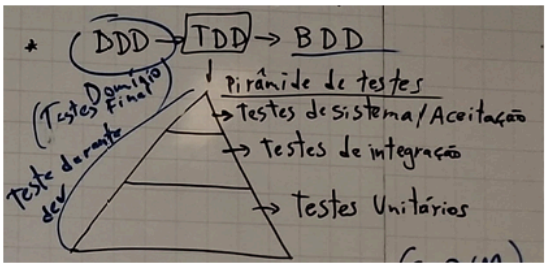
3. **Análise: DDD -> TDD -> BDD**

   Faça uma análise entre os processos DDD-> TDD-> BD

- DDD (Domain-Driven Design):
  - Foco no negócio, alinhando modelagem com requisitos.
- TDD (Test-Driven Development):
  - Testes escritos antes do código.
- BDD (Behavior-Driven Development):
  - Testes baseados em cenários para validar o comportamento.

IMAGENS IMPORTANTES:

- Testes de Software



  - Uso de BDD, metodologia (Scrum) comportamental, onde os testes são feitos antes do desenvolvimento, criando-se cenários para cada "feature" (Funcionalidade) usando os padrões "gherkin" (given.... when.... then.... call....)

- CI/CD e DevOps (Nuvem "Cloud)
  - CI = Integração Contínua
  - CD = Deploy Contínuo
  - DevOps = Metodologias para integrar e agregar às funções de desenvolvimento e produção contínua

  - Uso de pipelines (tubulações) ou sequências de atividades configuradas para automatizar o processo de entrega e produção

Padrões Criacionais

Os padrões criacionais tratam de como criar objetos de forma controlada e flexível.

1. **Factory Method**:
    - **O que é**: Define uma interface para criar objetos, mas permite que as subclasses decidam qual classe instanciar.
    - **Uso**: Quando o código precisa ser flexível ao criar objetos sem depender de suas classes concretas.
    - **Exemplo**: Um sistema de notificações que pode criar notificações por e-mail ou SMS dependendo da configuração.

2. **Abstract Factory**:
    - **O que é**: Fornece uma interface para criar famílias de objetos relacionados sem especificar suas classes concretas.
    - **Uso**: Útil quando diferentes implementações de um conjunto de objetos relacionados devem coexistir.
    - **Exemplo**: Criação de componentes gráficos para diferentes sistemas operacionais (Windows, macOS).

3. **Builder**:
    - **O que é**: Constrói objetos complexos passo a passo, separando o processo de construção de sua representação final.
    - **Uso**: Quando objetos têm muitas partes opcionais ou complexidade em sua criação.
    - **Exemplo**: Construção de um carro com diferentes configurações de motor, rodas e cor.

4. **Singleton**:
    - **O que é**: Garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a ela.
    - **Uso**: Quando é necessário controlar o acesso a um recurso único, como um logger ou conexão de banco de dados.
    - **Exemplo**: Uma única instância de um gerenciador de configuração.

5. **Prototype**:
    - **O que é**: Permite criar novos objetos copiando um protótipo existente.
    - **Uso**: Quando a criação de um objeto é cara ou complexa, e é mais fácil clonar um existente.
    - **Exemplo**: Clonagem de documentos ou configurações de usuário.

Padrões Estruturais

Os padrões estruturais focam na composição de classes e objetos para formar estruturas maiores.

1. **Adapter**:
    - **O que é**: Permite que interfaces incompatíveis trabalhem juntas.
    - **Uso**: Quando é necessário integrar classes ou APIs com interfaces diferentes.
    - **Exemplo**: Adaptar um sistema de pagamento antigo para uma nova API.

2. **Bridge**:
    - **O que é**: Desacopla uma abstração de sua implementação para que as duas possam variar independentemente.
    - **Uso**: Quando há múltiplas dimensões de variação em uma hierarquia de classes.
    - **Exemplo**: Dispositivos (TV, Rádio) controlados por diferentes controles remotos.

3. **Composite**:
    - **O que é**: Composição de objetos em estruturas de árvore para tratar elementos individuais e composições de forma uniforme.
    - **Uso**: Quando objetos compósitos precisam ser manipulados como objetos únicos.
    - **Exemplo**: Estrutura de arquivos em um sistema operacional.

4. **Decorator**:
    - **O que é**: Adiciona responsabilidades a um objeto dinamicamente, sem alterar sua estrutura.
    - **Uso**: Quando é necessário extender funcionalidades de maneira flexível.
    - **Exemplo**: Adicionar funcionalidades a janelas de uma interface gráfica (scroll, bordas, etc.).

5. **Facade**:
    - **O que é**: Fornece uma interface simplificada para um conjunto de interfaces em um subsistema.
    - **Uso**: Para reduzir a complexidade de um sistema ao expor apenas uma interface simples.
    - **Exemplo**: Um sistema de fachada para simplificar o acesso a um conjunto de APIs complexas.

6. **Flyweight**:
    - **O que é**: Reduz o consumo de memória compartilhando partes semelhantes de objetos entre instâncias.
    - **Uso**: Quando muitos objetos semelhantes precisam ser manipulados, mas o custo de memória é alto.
    - **Exemplo**: Representação de caracteres em um editor de texto.

7. **Proxy**:
    - **O que é**: Fornece um substituto ou marcador para controlar o acesso a outro objeto.
    - **Uso**: Para controlar acesso, otimizar chamadas ou fornecer funcionalidade adicional.
    - **Exemplo**: Proxy de segurança para controlar o acesso a recursos.

Padrões Comportamentais

Os padrões comportamentais lidam com a interação entre objetos e a delegação de responsabilidades.

1. **Observer**:
    - **O que é**: Define uma dependência um-para-muitos entre objetos, de forma que quando um objeto muda de estado, seus dependentes são notificados.
    - **Uso**: Quando múltiplos objetos precisam reagir a mudanças em outro objeto.
    - **Exemplo**: Um sistema de notificações de eventos.

2. **Strategy**:
    - **O que é**: Define uma família de algoritmos, encapsula cada um e os torna intercambiáveis.
    - **Uso**: Quando múltiplas variações de um algoritmo são necessárias.
    - **Exemplo**: Métodos de ordenação (bubble sort, quick sort).

3. **State**:
    - **O que é**: Permite que um objeto altere seu comportamento quando seu estado interno muda.
    - **Uso**: Quando o comportamento de um objeto depende de seu estado.
    - **Exemplo**: Um pedido de compra que muda de "em processamento" para "enviado" e, depois, "entregue".

4. **Chain of Responsibility**:
    - **O que é**: Encadeia objetos para que o pedido seja passado pela cadeia até que um objeto o processe.
    - **Uso**: Quando múltiplos objetos podem manipular um pedido, mas não é claro qual deles.
    - **Exemplo**: Sistema de suporte técnico que escalona chamadas.

5. **Command**:
    - **O que é**: Encapsula uma solicitação como um objeto, permitindo desfazer, refazer e armazenar operações.
    - **Uso**: Para implementar operações como histórico ou fila.
    - **Exemplo**: Funcionalidade de desfazer em editores de texto.

6. **Template Method**:
    - **O que é**: Define o esqueleto de um algoritmo na superclasse, permitindo que as subclasses implementem passos específicos.
    - **Uso**: Quando várias classes compartilham a estrutura básica de um algoritmo, mas têm variações específicas.
    - **Exemplo**: Processos de manipulação de arquivos em diferentes formatos.

7. **Visitor**:
    - **O que é**: Representa uma operação a ser realizada em elementos de uma estrutura de objetos.
    - **Uso**: Quando se deseja realizar novas operações em uma estrutura sem alterar suas classes.
    - **Exemplo**: Operações de impressão ou exportação em uma árvore de nós.

8. **Mediator**:
    - **O que é**: Define um objeto que encapsula como um conjunto de objetos interage.
    - **Uso**: Para reduzir o acoplamento entre objetos que se comunicam.
    - **Exemplo**: Sistema de chat onde o servidor gerencia as interações entre usuários.

9. **Memento**:
    - **O que é**: Captura e externaliza o estado interno de um objeto sem violar o encapsulamento.
    - **Uso**: Para implementar checkpoints ou desfazer operações.
    - **Exemplo**: Jogos que salvam o progresso do jogador.

10. **Interpreter**:
    - **O que é**: Fornece uma maneira de avaliar gramáticas ou expressões específicas.
    - **Uso**: Quando um problema envolve a interpretação de comandos ou linguagens.
    - **Exemplo**: Interpretadores de linguagens simples ou expressões matemáticas.