Aluno: Renato Augusto Platz Guimarães Neto

Em qual parte do projeto trabalhou? Qual relevância e aprendizado?

Minhas principais funções foram o estudo, escrita e desenvolvimento do Paradigma de Programação Orientada a Objetos, também fui um dos responsáveis em montar o slide para a apresentação.

Para cada apresentação descreva:
Houve relevância e/ou algo interessante? cite e justifique.

Como o paradigma de POO foi o que mais estudamos durante o curso, ao invés de focar em uma introdução ao mesmo, procurei trazer um maior aprofundamento ao paradigma. Para o desenvolvimento, decidi optar pela linguagem de programação multiparadigma e de propósito Geral C++, também conhecida como "C com classes", o principal motivo dessa escolha foi por ser uma linguagem que nasceu para ser orientada a objetos. A orientação a objetos nasce por uma necessidade, com seu criador sendo Alan Kay para a linguagem Smalltalk. Bjarne Stroustrup criou a linguagem C++ para acabar com a enorme complexidade dos códigos empresariais feitos em C, que chegava a ter de 25.000 a até 100.000 linhas de código, extremamente verbosos, falta de controle e péssima manutenibilidade e modularidade. A Orientação a Objetos é o paradigma mais popular, a maioria das linguagens de programação mais populares são preparadas para lidar com o paradigma, é obrigatório entender POO, principalmente quando se trabalha com sistemas mais complexos e robustos, como sistemas empresariais ou jogos digitais.

Apresentação Grupo “Enois”:
Análise de paradigmas de programação em módulos do VS Code;

Sim, o trabalho foi significativo, pois demonstra que não é preciso se limitar a um único paradigma para desenvolver um projeto, especialmente em uma IDE tão avançada e versátil como o VS Code. Dessa forma, compreendemos que é essencial explorar e aprender sobre diferentes abordagens de programação, mantendo-se sempre atualizado com novas metodologias e conhecimentos.

Apresentação do grupo “The Rapazes”:
Do paradigma ao resultado: Como diferentes abordagens resolvem os mesmos problemas de formas distintas

A equipe explora diversos paradigmas de programação e demonstra a refatoração de um código em um projeto real, comparando soluções alternativas. O estudo avalia a complexidade de implementação, a legibilidade do código e o esforço necessário para adaptá-lo a diferentes estilos de programação. A relevância está em evidenciar o custo de tempo e recursos envolvidos na refatoração para um novo

paradigma, ajudando a entender quando e por que vale a pena adotar uma abordagem diferente.

Apresentação do grupo “Los Bandoleiros”:
Comparação entre paradigmas: Orientado a Objetos e Imperativo

A equipe demonstra a implementação de um mesmo problema em dois paradigmas distintos: **Orientado a Objetos (POO)** e **Imperativo**. São analisados:

- **Legibilidade**: Como a estrutura do código (classes, objetos, funções ou procedimentos) influencia na clareza e organização.
- **Complexidade**: Comparação entre a abordagem modular (POO) e a linear (Imperativo), destacando vantagens e desafios em cada caso.
- **Principais diferenças**: Encapsulamento, reutilização de código e manutenibilidade no POO versus fluxo sequencial e simplicidade no Imperativo.

O objetivo é mostrar como cada paradigma aborda a solução de problemas, ajudando a escolher a melhor estratégia conforme o contexto do projeto.

Apresentação do grupo “YET”
Estudo sobre paradigma de programação Funcional

A equipe conduziu uma análise detalhada sobre o **Paradigma Funcional**, utilizando **Haskell** como referência para demonstrar seus princípios fundamentais e evolução histórica. O estudo incluiu:

- **Comparação com o Paradigma Imperativo**:
    - Diferenças na estruturação do código (funções puras vs. procedimentos com estado mutável).
    - Vantagens do funcional, como **imutabilidade**, **facilidade de teste** e **raciocínio matemático**.
- **Exemplos Práticos**:
    - Demonstração de soluções para o mesmo problema em ambos paradigmas, destacando clareza e eficiência.
- **Classificação de Linguagens Funcionais**:
    - **Puras** (ex: Haskell, Elm).
    - **Fortemente funcionais** (ex: Scala, F#).
    - **Parcialmente funcionais** (ex: JavaScript, Python).

Aplicações do Paradigma Funcional

Indicado para cenários que demandam:

- **Confiabilidade**: Sistemas distribuídos (telecomunicações, roteamento).
- **Processamento de Dados**: *Machine Learning* e *NLP* (Haskell, Erlang).
- **Front-end Moderno**: Bibliotecas como **React.js** e **Redux**, que adotam imutabilidade e composição de funções.

Conclusão: Embora exija adaptação, o paradigma funcional oferece vantagens em projetos que priorizam **manutenibilidade**, **escalabilidade** e **baixa propensão a erros**.

Apresentação do grupo "CKS":
Estudo do Paradigma de Programação Orientado a Objetos

O grupo explorou a base matemática do paradigma funcional, fundamentada no conceito de **funções declarativas**, utilizando **Java** como linguagem de estudo — uma linguagem parcialmente funcional que incorpora esses princípios através de:

- **Expressões lambda**
- **Stream API** (operações como `map`, `filter`, `reduce`)
- **Interfaces funcionais** (ex: `Function<T,R>`, `Predicate<T>`)

Vantagens do Paradigma Funcional em Java

1. **Previsibilidade**: Funções puras garantem que a saída dependa exclusivamente dos parâmetros de entrada, evitando efeitos colaterais.
2. **Concorrência mais segura**: Imutabilidade simplifica o trabalho com threads.
3. **Código conciso**: Operações como `filter` e `map` substituem loops verbosos.

Desafios e Desvantagens

1. **Curva de aprendizado íngreme**: Exige mudança de mentalidade, especialmente para desenvolvedores acostumados ao paradigma imperativo/OO.
2. **Abordagem distinta para resolução de problemas**: Necessidade de pensar em termos de composição de funções, imutabilidade e ausência de estado.
3. **Ecossistema Java majoritariamente imperativo**:
    - Bibliotecas e frameworks tradicionais (ex: Spring) foram projetados para OO.
    - Integração de códigos funcionais e imperativos pode gerar inconsistências.

Conclusão

Embora o Java permita adotar conceitos funcionais, sua natureza híbrida exige cuidados. O paradigma funcional traz benefícios em **legibilidade** e **manutenibilidade**, mas sua adoção total esbarra na cultura predominante de OO na comunidade Java. Ideal para cenários específicos, como processamento de coleções ou operações paralelas, mas não substitui completamente outras abordagens.