Qual code smell mais apareceu em seus projetos?

- Long Method

- Long Method

- Long metodh e code class

- Long Method e consequentemente God Class.

- God Class. A classe Tabuleiro fazia tudo sozinha, desde a lógica até o método main

- Long Methods e Large Classes

- Multiplicidade de responsabilidades para determinado metodo de uma classe

- long method

- God class e Long Method

- O mais evidente e que mais apareceu foi a falta de validação nos métodos e nas demais partes do código. O meu código não estava bem-preparado para eventuais erros de digitação e interpretação do usuário.

- Long method

- Long Statements ou Long Methods

- Long method

- Long metodh

- long methods

Quais métricas de código foram mais negativas em seus projetos? (máximo de 3)

- Duplicação de código
Grande quantidade de issues

- Manutenibilidade

- Modularidade e complexidade

- Tamanho, complexidade e de coesão

- Complexidade do método principal, Encapsulamento e Responsabilidade única (já que tudo estava no main)

- quantitativas, qualitativas e discretas

- Acoplamento e Manutenibilidade

- manutenibilidade

- Coesão e Acoplamento

- Métodos longos foi a métrica mais negativa, e temos como exemplo o método jogo.start(), que seria o responsável por todo o fluxo do jogo.

- Complexidade,Modularidade

- uso da função Random

- uso de System.out.println

- Manteinabilidade

- Manutenibilidade e confiabilidade

- manutenibilidade e padrões de código

Quais refatorações se mostraram as mais aplicáveis em seus projetos? (máximo de 3)

- Refatorar grandes métodos, dividindo eles em metodos menores; Utilização do padrão de projeto Facade que melhorou a classe Main

- Convenções de nomes de pacotes, variáveis ou classes, correção de más práticas e correção de métodos inutilizados.

- Obsessão por primitivo e inveja de recursos

- Extract Method e Extract Class

- Extração de Classes, Substituição de Condicionais por Polimorfismo e Encapsulamento

- legibilidade, manutenção e desempenho

- A divisão de um método em diversos pra diminuir a complexidade e a delegação de responsabilidades pra classes

- não lembro

- Extract Function, Extract Variable e Inline Function

- A primeira refatoração foi a mudança no jogo.start(), no qual quebrei o código em partes que seriam outros métodos privados, para que o código não ficasse 100% concentrado no jogo.start()

- O princípio da responsabilidade única,Encapsulamento,nomes significativos

- convenções de nomes de pacotes, variáveis ou classes

- Separar as responsabilidades em classes

- Separar as responsabilidades das respectivas classes e renomear métodos

- encapsular campos e especializar métodos

Quais problemas nos seus projetos podem ser resolvidos com a aplicação de padrões de software?

- A grande complexidade da classe Main (Facade);
Melhor utilização da hierarquia de classes abstratas (Structure)

- Organização e reutilização de código a longo prazo.

- Um problema básico é a separação de funções no código, que sem a aplicação de métodos pode gerar uma complicação no próprio

- difícil interpretação e perda desnecessária de tempo

- Meu código tinha muitas responsabilidades em uma única classe. O uso do padrão facade ajudou a centralizar a configuração e execução do sistema na classe Jogo, reduzindo a complexidade do main e separando melhor as responsabilidades

O padrão Strategy permitiu encapsular as regras das diferentes casas do jogo em classes específicas, facilitando a manutenção e a adição de novos comportamentos

Já o padrão Factory resolveu o problema da criação manual e repetitiva de objetos, deixando o código mais flexível, limpo e organizado

- facilidade de manutenção para adicionar novos atributos no código

- Duplicação de código, delegação de responsabilidades mais assertivas pra classes como a factory para criação de objetos, e o alto nível de acoplamento tbm é um problema facilmente resolvido com padrões de software como o Chain of Responsability

- duplicação de codigo e classes com muitas responsabilidades

- Diminuição de duplicações e maior Manutenibilidade

- Problemas como a lógica complexa de if-else para as regras das casas, são resolvidos com o padrão Strategy, que encapsula cada regra numa classe própria. A criação confusa de diferentes tipos de jogadores é simplificada pela Factory, que centraliza essa lógica. A necessidade de ter um único tabuleiro acessível por todo o sistema é resolvida pelo Singleton.

- Código duplicado e difícil de manter e facilitou a implementação da interface

- A maior parte dos code smells, entre outros problemas, foram resolvidos ao usar padrões de software

- Código difícil de manter

- Manutenção do código

- manutenção de código

Acerca dos conteúdos discutidos (code smells, métricas, código limpo e padrões), quais foram suas maiores dificuldades?

- Idênticas os code smells sem usar as plataformas adequadas para isso

- Entender e implementar os padrões

- Entende o porqie de se aplicar e o porque disso ocorrer de forma inerente a quem escreve.

- code smells, por ter sido o primeiro dos conteúdos.

- tive um pouco de dificuldade em aplicar os padroes Strategy, Facade e Factory.

- code smell

- Abstrair os conceitos pra aplicações mais robustas de nível comercial visto que o nível trabalhado no semestre passado era pra aplicações mais simples, porém, não foi um ponto negativo pois creio que nos proporcionou uma melhor visão de campo na área pra aplicações futuras de maior qualidade.

- entender como implementar os padrões e pensar antes de fazer o codigo

- Code Smells

- Implementar o código limpo e os padrões.

- aplicar o padrao de projeto strategy, tentar aplicar o principio do aberto e fechado, e refatorar o codigo do jogo do tabuleiro

- compreender os padrões em um curto período de tempo para realização do trabalho prático

- Decorar os padrões e abstrai-los

- Code smells e padrões de software

- código limpo e padrões

Acerca dos conteúdos discutidos (code smells, métricas, código limpo e padrões), o que você achou mais relevante / importante?

- O uso no SonarQube para avaliar as métricas e os code smells do código e a utilização dos padrões que deixa todo o projeto mais limpo e estruturado

- Todos.

- A forma de como aplicar elas e de como elas podem ser alteradas de formas simples para cumprir o que se precisa referente a elas.

- padrões, por ser o mais prático dentre eles e mostrar formas de melhorar o código.

- Achei os padrões de projeto os mais relevantes, pois meu código estava desorganizado e confuso. Aplicar os padrões me ajudou a estruturar melhor as ideias e tornou meu projeto mais fácil de entender, tanto para mim quanto para os outros

- code smell

- Padrões de Projetos, pois com ele conseguimos montar projetos inteiros de maneira limpa e organizada

- padrões

- Padrões de projeto, pois ajudam a organizar melhor o projeto.

- Code smells e métricas são muito úteis para achar possíveis dores de cabeça, enquanto o código limpo facilita a leitura do código por diferentes desenvolvedores, mas os padrões são o conteúdo mais relevante.

- Código limpo é o mais importante, pois serve de base para identificar code smells, aplicar padrões corretamente e interpretar métricas de forma eficaz.

- padrões e smells

- código limpo

- Padrões de software e código limpo

- code smells e padrões

Descreva, brevemente, o que você achou da disciplina de POO englobar aulas e trabalhos sobre o conteúdo envolvendo qualidade de software e a forma como esses conceitos foram apresentados.

- Foi algo essencial, já que utilizaremos a grande maioria desses conceitos em todos os futuros projetos e códigos

- Muito interessante e útil, bem apresentado.

- Achei ótima, pois são conceitos de extrema importância e que quanto antes sabermos como eles são e de como aplicar eles se torna melhor a experiência pra código futuros também tanto code smells como padrões de projetos.

- achei importante. é um conhecimento necessário para programadores. me ajudou a ter uma mentalidade de questionar sempre se o código está o mais legível possível para o próximo colega que for lê-lo. e os trabalhos reforçam esse conhecimento, fixando o conteúdo e garantindo que seja absorvido.

- Gostei bastante, especialmente dos padrões de projeto, que junto com o estudo de code smells ajudam a identificar erros e melhorar a programação. Isso me ajudou a entender melhor como escrever um código mais organizado

- melhorou organizar melhor meu código

- Como dito anteriormente, ainda que este seja um assunto mais avançado, acho que cabe perfeitamente com a disciplina e é de extrema importância para a formação do aluno

- Excelente

- Achei positiva a inclusão de conteúdos sobre qualidade de software na disciplina de Programação Orientada a Objetos (POO). Ao abordar refatoração e padrões de projeto, as aulas e os trabalhos ajudaram a ir além da simples implementação de código, m,ostrando como desenvolver um projeto cada vez mais profissional e organizado.

- Foi uma experiência muito boa para mim que pretendo entrar na carreira de dev, pois aprendi conceitos muito relevantes para melhorar a qualidade do meu código e também para me engajar no mercado de trabalho.

- Achei muito positivo ,pois isso amplia a visão do aluno além da programação em si, mostrando a importância de escrever código legível, reutilizável e de fácil manutenção.

- achei muito relevante, principalmente considerando cadeiras futuras

- Achei interessante, é uma coisa que agrega para o futuro como um bom desenvolvedor

- Achei muito importante, pois engloba conceitos importantes que vão agregar na minha carreira como um bom desenvolvedor

- foi essencial para nosso desenvolvimento profissional, já que o conteúdo, na maioria da vezes, não é abordado neste momento da faculdade. Isso nos permite melhorar projetos mais cedo e evoluir mais rapidamente.