# Padrões e Práticas de Desenvolvimento

2022/10

# Agenda

- ❑ Clean Code
- ❑ Qualidade do código
- ❑ Code Smells
- ❑ Refactoring
- ❑ Testes
- ❑ Mãos na massa
- ❑ Tira dúvidas

# Afinal, o que é Clean Code?

## Clean Code

“Código limpo sempre parece que foi escrito por alguém que se importa”

Michael Feathers
(Autor do livro Working Effectively with Legacy Code)

## Clean Code

“Você sabe que está trabalhando em um código limpo quando cada rotina que você lê acaba sendo praticamente o que você esperava”

Ward Cunningham (Criador da Wiki)

## Clean Code

“Qualquer um pode escrever código que um computador entende. Bons programadores escrevem código que humanos podem entender”

Martin Fowler
(Um dos maiores autores sobre desenvolvimento de software)

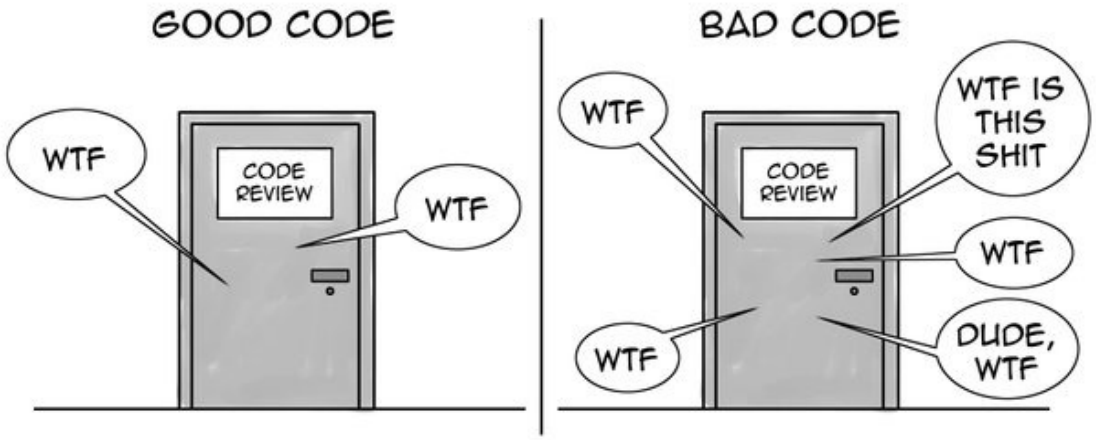
## Clean Code

“Clean Code, ou Código Limpo, é uma filosofia de desenvolvimento de softwares que consiste em aplicar técnicas simples que facilitam a escrita e a leitura de um código. Tornando-o, assim, de fácil compreensão”

# Como medir a qualidade do código?

# Linhas de código?
# Número de métodos?
# Número de classes?
# Tamanho dos métodos?
# Complexidade?

# Qualidade do código



# Qualidade do código

```
void quickSort(int[] valor, int esquerda, int direita)
{
    int i, j, x, y;
    i = esquerda;
    j = direita;
    x = valor[(esquerda + direita) / 2];

    while(i <= j)
    {
        while(valor[i] < x && i < direita)
        {
            i++;
        }
        while(valor[j] > x && j > esquerda)
        {
            j--;
        }
        if(i <= j)
        {
            y = valor[i];
            valor[i] = valor[j];
            valor[j] = y;
            i++;
            j--;
        }
    }
    if(j > esquerda)
    {
        quickSort(valor, esquerda, j);
    }
    if(i < direita)
    {
        quickSort(valor,  i, direita);
    }
}
```

```
41  @   static MappedField validateQuery(final Class clazz, final Mapper mapper, final StringBuilder origProp, final FilterOperator op, final
42          MappedField mf = null;
43          final String prop = origProp.toString();
44          boolean hasTranslations = false;
45          if (!origProp.substring(0, 1).equals("$")) {
46              final String[] parts = prop.split( regex: "\\.");
47              if (clazz == null) { return null; }
48              MappedClass mc = mapper.getMappedClass(clazz);
49              //CHECKSTYLE:OFF
50              for (int i = 0; ; ) {
51                  //CHECKSTYLE:ON
52                  final String part = parts[i];
53                  boolean fieldIsArrayOperator = part.equals("$");
54                  mf = mc.getMappedField(part);
55                  //translate from java field name to stored field name
56                  if (mf == null && !fieldIsArrayOperator) {
57                      mf = mc.getMappedFieldByJavaField(part);
58                      if (validateNames && mf == null) {
59                          throw new ValidationException(format("The field '%s' could not be found in '%s' while validating - %s; if you wi
60                      }
61                      hasTranslations = true;
62                      if (mf != null) {
63                          parts[i] = mf.getNameToStore();
64                      }
65                  }
66                  i++;
67                  if (mf != null && mf.isMap()) {
68                      //skip the map key validation, and move to the next part
69                      i++;
70                  }
71                  if (i >= parts.length) {
72                      break;
73                  }
74                  if (!fieldIsArrayOperator) {
75                      //catch people trying to search/update into @Reference/@Serialized fields
76                      if (validateNames && !canQueryPast(mf)) {
77                          throw new ValidationException(format("Cannot use dot-notation past '%s' in '%s'; found while validating - %s", pa
78                      }
79                      if (mf == null && mc.isInterface()) {
80                          break;
81                      } else if (mf == null) {
82                          throw new ValidationException(format("The field '%s' could not be found in '%s'", prop, mc.getClazz().getName())
83                      }
84                      //get the next MappedClass for the next field validation
85                      mc = mapper.getMappedClass((mf.isSingleValue()) ? mf.getType() : mf.getSubClass());
86                  }
87              }
88
89              //record new property string if there has been a translation to any part
90              if (hasTranslations) {
91                  origProp.setLength(0); // clear existing content
92                  origProp.append(parts[0]);
93                  for (int i = 1; i < parts.length; i++) {
```

What's a prop?

What's a part?

Eeeh!

Why all the null checks?

Control the loop

Comments, because code is unclear

Parameter mutation!

## Code Smells

# O que significa code smells?

## Code Smells

“Code Smell (cheiro de código) é qualquer característica no código-fonte de um programa que possivelmente indica um problema ou uma má pratica desenvolvimento.”

## Code Smells

# Os Code Smells mais comuns:

- ❑ Bloaters
- ❑ Object-Orientation Abusers
- ❑ Change Preventers
- ❑ Dispensables
- ❑ Couplers

Code Smells

# Bloaters

Bloaters são códigos, métodos e classes que aumentaram para proporções tão gigantescas que são difíceis de trabalhar. Normalmente, esses smells não surgem imediatamente, mas se acumulam ao longo do tempo à medida que o programa evolui (e especialmente quando ninguém faz um esforço para erradicá-los).

Code Smells

# Object-Orientation Abusers

Trata-se da violação à orientação de objetos. Assim, a aplicação inadequada total ou parcialmente da herança, polimorfismo ou encapsulamento podem provocar problemas no código. Para identificar este problema, repare se o código tem muitos ifs ou switches complexos, classes com ações idênticas somente com nomes diferentes, atributos e métodos criados porem sem usos.

Code Smells

# Change Preventers

É o tipo de código que dificulta a manutenção, pois está relacionado à ideia de “código espaguete”. Ele é aquele que ao “puxar” um ponto para correção, automaticamente se atinge outros pontos. De maneira geral, códigos assim devem ser reescritos, separados, isolados ou ate apagados pois estão sem uso.

Code Smells

# Dispensables

São os trechos do código que podem ser removidos sem afetar a aplicação, como comentários longos, partes duplicadas e outros itens que deixam o código-fonte longo demais. Afinal, um código enxuto reduz a chance de erros e falhas.

Code Smells

# Couplers

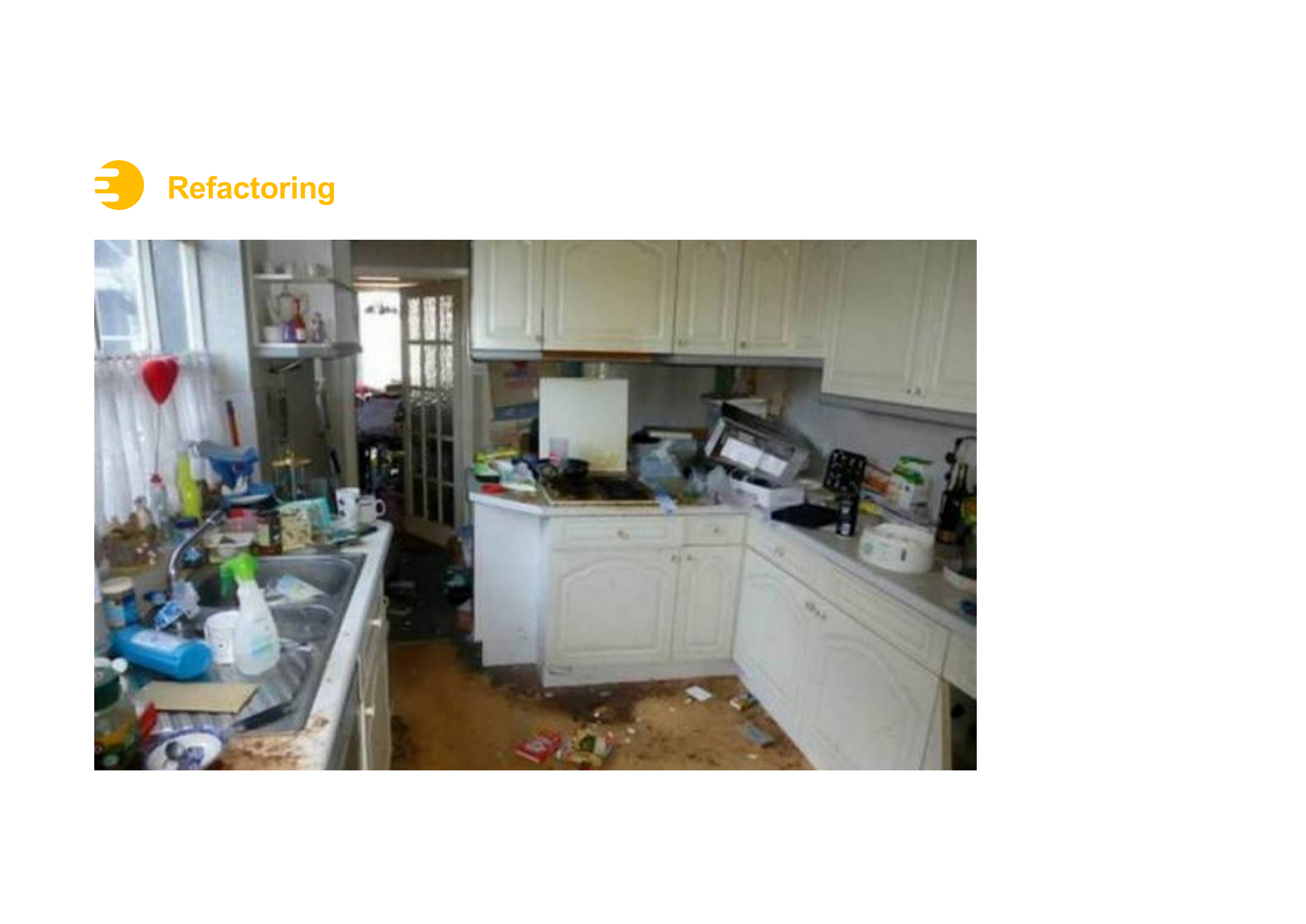
Trata-se do alto acoplamento, que é a forma que temos de medir o quão dependente uma classe é das outras. Assim, uma boa prática da Programação Orientada a Objetos (POO) é que uma aplicação de qualidade tem baixo acoplamento. Portanto, o ideal é prevenir este problema.

# O que é refactoring?

## Refactoring

“Alteração feita na estrutura interna do software para torna-lo mais fácil de ser entendido e menos custoso de ser modificado, sem alterar o seu comportamento observável”

Martin Fowler


Refactoring

## Refactoring

- A refatoração deve ser encarado como um investimento, pois torna o software sustentável e competitivo
- Dia após dia a falta de refatoração irá consumir cada vez mais o tempo da equipe
- Aproveite a hora adicionar novas funcionalidades para refatorar coisas relacionadas
- Refatore quando for corrigir um defeito, esta é uma ótima oportunidade
- Refatore após precisar entender uma parte do código complexa, pois pode ser que precise voltar nela em um futuro próximo
- Refatore sempre!

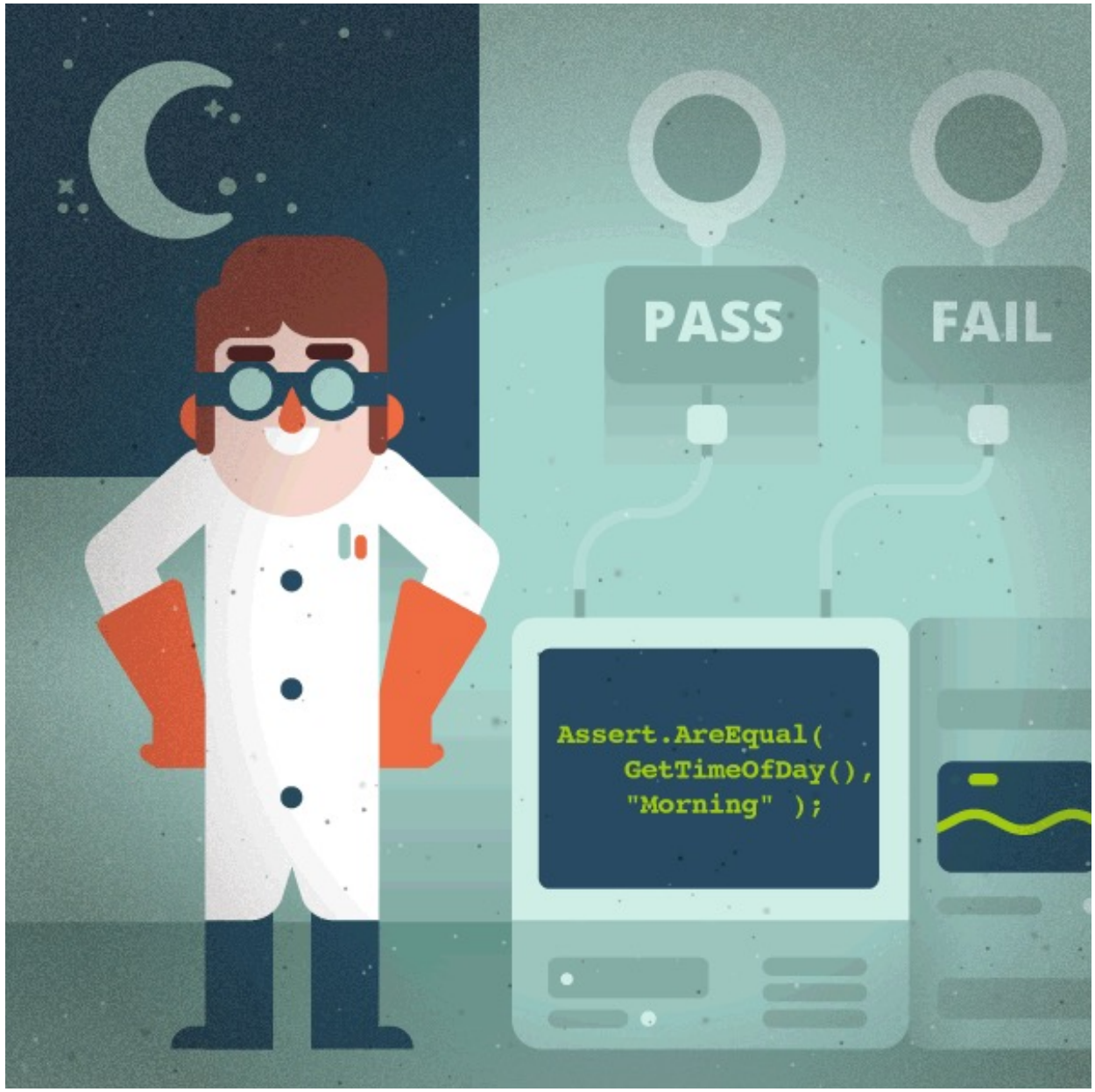
Refactoring

# Total Productive Maintenance (TPM)

1. **Ordenação:** nomear variáveis, funções e arquivos de forma apropriada para que você consiga encontrar o que está procurando.
2. **Sistematizar:** manter cada coisa em seu lugar. Um trecho de código deve estar onde você espera encontrar, caso não esteja, mova para o local certo.
3. **Limpeza:** manter o seu código limpo, indentar corretamente, remover blocos antigos de código e que foram comentados.
4. **Padronização:** seguir um padrão, em que todo o time faz as coisas da mesma forma, no mesmo estilo.
5. **Disciplina:** mantenha disciplina, siga as regras e esteja aberto a mudanças pessoais para se adaptar ao que foi combinado.

Testes

# Para que testes?



## Testes

“Os testes não nos garante que tenhamos um sistema 100% sem falhas, porem nos traz uma tranquilidade e segurança na hora da manutenção do código, possibilitando que de uma forma rápida e eficaz identifiquemos uma possível falha e assim resolve-la, sem eles ficamos torcendo para que o sistema simplesmente funcione”

Testes

# Estrutura de um teste automatizado

**Given:** Definição de todas as informações necessárias para executar o comportamento que será testado
**When:** Executar o comportamento
**Then:** Verificar o que aconteceu após a execução, comparando as informações retornadas com a expectativa que foi criada

Testes

# Exemplo, cancelar um pedido existente:

**Dado** um pedido existente, contendo apenas o código
**Quando** o pedido for cancelado
**Então** deve ser retornado uma confirmação do cancelamento do pedido, caso ele ainda não tenha sido pago, além disso os itens devem ser liberados no estoque

# Tipos de teste automatizado:

# Mãos na massa!!!

Tira Duvidas

# Temos Duvidas???

## O que vem por ai?

Os assuntos abordados aqui, devido ao tempo limitado, foram apresentados de forma resumida, possibilitando que apresentássemos uma maior quantidade de conteúdo, porém os mesmos devem ser pesquisados e aprofundados para uma melhor empregabilidade no seu dia a dia. Além dos assuntos que aqui foram apresentados, recomendamos abaixo alguns outros que estão cada dia mais sendo requisitados em entrevistas de empresas:

- ✓ Design Patterns
- ✓ SOLID
- ✓ TDD
- ✓ Clean Architecture
- ✓ DevOps

eduzz

Obrig@**duzz !**

**Th@nk you !**

# Referências


https://brasap.com.br/o-que-e-um-codigo-bom-e-um-codigo-ruim-e-o-que-diferencia-os-estilos-de-codificacao/
https://coodesh.com/blog/dicionario/o-que-e-code-smell/
https://balta.io/artigos/clean-code
https://refactoring.guru/pt-br/refactoring/smells#:~:text=Bloaters,an%20effort%20to%20eradicate%20them).
https://www.altexsoft.com/blog/engineering/code-refactoring-best-practices-when-and-when-not-to-do-it/
https://www.macoratti.net/alg_cpf.htm
https://medium.com/leti-pires/c%C3%B3digo-limpo-parte-01-ffcb90215b4a
https://www.google.com/