LABORATÓRIOS DE PROGRAMAÇÃO

Código Limpo

Aula 2

Profª. Kamilla Dória

TÓPICOS

- Introdução
- Recomendações

INTRODUÇÃO

CÓDIGO LIMPO

- É um estilo de desenvolvimento que tem por foco a facilidade para escrever, ler e manter o código.

- Robert C. Martin, em seu livro “Clean Code: A Handbook of Agile Software Craftsmanship”, afirma que a proporção de leitura para escrita do código é de 10:1. Por isso, um código bem escrito que facilite a leitura é não só desejável, mas necessário no cenário atual.

CÓDIGO LIMPO

- Um software nunca está finalizado!
- Sempre há a necessidade de atualizações ou novas funcionalidades.

CÓDIGO LIMPO

- Caso o software não esteja com um código limpo, o desenvolvimento e manutenção se tornarão cada vez mais complicados.

- Até chegar ao ponto em que será mais simples começar o software do zero do que prosseguir com um código ruim.

CÓDIGO CONFUSO

- Pegar um código confuso de outra pessoa (ou até um antigo código seu) pode causar baixa produtividade.

- Equipes que trabalharam rapidamente no início do projeto, após 2 anos não possuem mais a mesma produtividade;

  - Cada alteração causa falha em outra parte do código;

  - Nenhuma alteração é trivial;

  - Cada adição ou modificação necessita de um entendimento anterior.

CÓDIGO CONFUSO

- Equipes que trabalharam rapidamente no início do projeto, após 2 anos não possuem mais a mesma produtividade;
  - Conforme a confusão aumenta, a produtividade cai;
  - Adicionar pessoas aumenta ainda mais a confusão.

GRANDE REPLANEJAMENTO

- Em algum momento da confusão do código a equipe clama por replanejamento, pois não suportam mais trabalhar com um código irritante;

- Alguns partem para construir um novo sistema em uma outra equipe;

  - Riscos:

    - Acompanhar as novas funcionalidades no sistema antigo e implementar no novo;

    - Membros da equipe são substituídos e mais tarde pedem novo replanejamento.

ATITUDE

- “A culpa é do chefe..”.
    - O programador deve proteger o código!

RECOMENDAÇÕES

NOMES

- Nomes são importantes!

- É comum as pessoas serem apelidadas pelas suas características mais comuns. Para termos um código limpo, devemos fazer isso com ele.

  - Nomear variáveis, funções, parâmetros, classes ou métodos de acordo com suas funcionalidades.

NOMES

- Ao definir um nome, precisamos ter em mente dois pontos principais:
  - Ser preciso;
    - Precisamos passar a ideia central da nossa variável ou método, sem dar voltas, sendo conciso e direto;
  - Não ter medo de nomes grandes;
    - Um nome bem descritivo, mesmo que seja grande, irá possibilitar uma melhor compreensão e posterior manutenção do código.

NOMES

- Evite abreviações;
  - Em muitos casos procuramos praticidade, e junto com uma ideia de negócio ou algo simplesmente que se tornou recorrente, denominamos abreviações para agilizar nossa escrita;
  - Dificultam entendimentos de negócio para novos membros do time ou até simplificam contextos que não deveriam ser simplificados ou menosprezados;
  - Seu uso é algo muito fácil de se tornar uma rotina e se alastra rapidamente pelo código de todo o projeto.

NOMES

- Passe a ideia central;
  - Identificar a ideia central e a intenção de um contexto pode ser um grande exercício de reflexão que pode nos trazer uma evolução constante.
  - O ato de nomear uma variável, método ou classe é algo prático e simples que podemos utilizar diariamente.

NOMES

- Isole as palavras de referências de negócio;
  - Palavras de referências de negócio são praticamente inevitáveis dentro de um projeto, mas procurar evitá-la ou isolá-la pode ser uma prática importante que irá facilitar o desacoplamento e permitirá abstrair mais casos em seu código;
  - Procure achar uma referência mais genérica do mundo externo ao negócio e especialize o que precisar.

NOMES

- Recomendado:
  - Métodos ou Funções:
    - Devem ter nome de verbos (gosto bastante de utilizar do infinitivo), para assim, expressar quais são suas finalidades;
  - Classes e Objetos:
    - Deve ser utilizado substantivos.

Don't Repeat Yourself

- Criado por Andy Hunt;
- Procura reduzir a duplicação de código e os problemas oriundos dessa prática;
- Definição formal:
  - "Cada parte do conhecimento deve ter uma representação única, não ambígua e definitiva dentro do sistema."

Don't Repeat Yourself

- Quando você estiver desenvolvendo e desconfiar que está usando um código semelhante (duplicado) mais de uma vez, pense um pouco no que aquilo significa para o sistema e decida de que forma faz mais sentido ele ser implementado e exposto adequadamente.

DRY

Don't Repeat Yourself

- Alguns fatores que devemos considerar quando estamos desenvolvendo. O código deve ser:
  - Fácil de se manter, adaptar e se ajustar às alterações de escopo;
  - Testável e de fácil entendimento;
  - Extensível para alterações com o menor esforço necessário;
  - Que forneça o máximo de reaproveitamento;
  - Que permaneça o máximo de tempo possível em utilização.

COMENTÁRIOS

- Comente o necessário e somente o necessário!
  - Códigos são constantemente modificados, enquanto comentários, raramente. Assim, é comum um comentário deixar de ter significado, ou pior ainda, passar um significado falso depois de algum tempo.
  - Além disso, códigos com muitos comentários são tão ruidosos que, com o tempo, nossos olhos acabam ignorando todos.

TRATAMENTO DE EXCEÇÕES

- Bons desenvolvedores pensam que as coisas podem dar errado, pois isso eventualmente irá acontecer. Desta forma, o código deve estar preparado para lidar com esses problemas que surgirão.
  - Exceptions:
    - Mecanismo que sinaliza eventos excepcionais. Por exemplo, tentar inserir o caractere “a” em uma variável do tipo inteiro;
  - Blocos try-catch:
    - Capturam as exceções citadas. Portanto, devem ser utilizados de maneira global.

REGRA DO ESCOTEIRO

- “Deixe o acampamento mais limpo do que você o encontrou.”
- Para desenvolvedores, podemos adaptar para:
  - “Deixe o código mais limpo do que estava antes de você mexer nele.”

REGRA DO ESCOTEIRO

- Pensamentos como “isso é perder tempo valioso” ou “se tá funcionando melhor não mexer” são prejudiciais a longo prazo.

- O melhor é arrumar o mais rápido possível enquanto a lógica ainda está fresca na cabeça.

TESTES LIMPOS

- Um código só está realmente limpo se ele for validado.

- Da mesma maneira que mantemos o nosso código limpo, com clareza, simplicidade e consistência de expressão, mantemos nossos testes limpos.

TESTES LIMPOS

- Seguem as regras do acrônimo FIRST (Fast, Indepedent, Repeatable, Self-validation, Timely).
  - Rapidez:
    - Os testes devem ser rápidos para que possam ser executados diversas vezes;
  - Independência:
    - Quando testes são dependentes, uma falha pode causar um efeito dominó dificultando a análise individual;

TESTES LIMPOS

- Seguem as regras do acrônimo FIRST (Fast, Indepedent, Repeatable, Self-validation, Timely).
  - Repetitividade:
    - Deve ser possível repetir o teste em qualquer ambiente;
  - Auto validação:
    - Bons testes possuem como resultado respostas do tipo “verdadeiro” ou “falso”. Caso contrário, a falha pode se tornar subjetiva;

TESTES LIMPOS

- Seguem as regras do acrônimo FIRST (Fast, Indepedent, Repeatable, Self-validation, Timely).
    - Pontualidade:
        - Os testes precisam ser escritos antes do código de produção, onde os testes serão aplicados. Caso contrário, o código pode ficar complexo demais para ser testado ou até pode ser que o código não possa ser testado.

EXERCÍCIO 1

A partir das recomendações vistas em sala de aula, melhore seu código produzido no exercício 1 da aula anterior.

Marque as alterações realizadas por meio de comentários.

EXERCÍCIO 1

Você está iniciando um novo negócio: uma locadora de veículos. Para iniciar o controle do aluguel de carros é necessário um sistema simples. Dessa forma, desenvolva um programa um Python para:

a) Solicitar ao usuário informações sobre as locações: nome do cliente, sexo (F- Feminino, M - Masculino), placa do carro alugado, quantidade de quilômetros contratados, quantidade de dias contratados;

b) Calcular e imprimir a placa do carro e valor total a pagar para CADA cliente, considerando que deverá ser cobrado o valor de R$ 70,00 por dia contratado, e R$ 0,10 para cada quilômetro contratado;

c) Calcular e imprimir a média de quilômetros contratados pelos clientes;

d) Calcular e imprimir o nome das clientes de sexo feminino que fecharam aluguéis acima de 7 dias contratados.

Obs.: o programa encerra quando o usuário informa o texto SAIR.

EXERCÍCIO 2

A partir das recomendações vistas em sala de aula, melhore seu código produzido no exercício 2 da aula anterior.

Marque as alterações realizadas por meio de comentários.

EXERCÍCIO 2

A LOTOFACIL consiste na extração de 15 números aleatórios diferentes, no universo de 01 a 25. Você marca entre 15 a 18 números, dentre os 25 disponíveis no volante, e fatura o prêmio se acertar 11, 12, 13, 14 ou 15 números. Pode ainda deixar que o sistema escolha os números para você por meio da Surpresinha. Considerando estas informações, faça um programa em Python para:

a) Solicitar ao usuário a quantidade de dezenas que ele deseja marcar na primeira aposta (entre 15 e 18 números). Caso o usuário informe uma quantidade de dezenas fora do intervalo válido, o programa deve solicitar nova digitação, tantas vezes quantas forem necessárias;

b) Solicitar ao usuário informar os números da primeira aposta (dezenas de 01 a 25, sem repetição). Caso o usuário informe um número repetido, o programa deverá apresentar uma mensagem "Número repetido" e solicitar nova digitação. Assim como se o usuário informar um número fora do intervalo válido, o programa deverá apresentar uma mensagem "Dezena inválida" e solicitar nova digitação.

c) Gerar aleatoriamente duas apostas, com 18 números, usando a "Surpresinha".

d) Simular o resultado (15 dezenas sorteadas) de um concurso da Lotofácil;

e) Imprimir (em ordem crescente) as dezenas da primeira aposta, das duas apostas (surpresinha) e do resultado do concurso da Lotofácil simulado.

OBRIGADA!