Código Limpo:

Habilidades Práticas do Agile Software

•••

Capítulo 2: Nomes Significativos

Este material foi desenvolvido pelo grupo PET/ADS do IFSP São Carlos

Use nomes que revelem seu propósito

Escolher o nome de uma variável leva tempo, mas economiza mais. Portanto, cuide dos nomes e troque-os quando achar melhores.

O nome de uma variável, função ou classe deve responder porque ela existe, o que faz e como é usada.

Se um nome requer um comentário que o explique, então ele não revela o seu propósito e deve ser trocado. Veja um exemplo:

```
int d; // tempo decorrido em dias
```

```
int elapsedTimeInDays;
```

Use nomes que revelem seu propósito: exemplo

Qual o propósito do código a seguir?

```
public List<int[]> getThem(){
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
       if (x[0] == 4)
          list1.add(x);
   return list1;
}
```

Que tipos de coisa estão em theList?

Qual a importância de um item na posição zero na theList?

Qual a importância do valor 4?

Como usar a lista retornada?

Use nomes que revelem seu propósito: exemplo melhorado

As respostas poderiam estar no código:

```
public List<int[]> getFlaggedCells(){
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for(int[] cell: gameBoard)
     if (cell[STATUS_VALUE] == FLAGGED)
         flaggedCells.add(cell);
     return flaggedCells;
}
```

O domínio do problema é um jogo de campo minado e board é uma lista de células. Logo, vamos chamá-la de gameBoard.

Cada quadrado do tabuleiro representa um vetor e a posição zero armazena o status. O valor 4 significa "marcado com uma bandeirinha".

Use nomes que revelem seu propósito: exemplo versão final

Uma classe poderia representar Células:

```
public List<Cell> getFlaggedCells() {
   List<Cell> flaggedCells = new ArrayList<Cell>();
   for( Cell cell : gameBoard)
      if (cell.isFlagged())
         flaggedCells.add(cell);
   return flaggedCells;
}
```

O código ainda é simples, porém muito mais explícito.

Com a classe criada, colocamos uma “flag” para denominar a função da célula em questão.

Evite informações Erradas

Nomes imprecisos e enganosos podem levar a confusão e erros, dificultando a manutenção e compreensão do código.

Desenvolvedores devem escolher nomes claros, descritivos e precisos para suas variáveis, funções e classes.

```
int accountList; //variável com o nome List, não sendo uma list
```

```
int accountGroup; //mesma variável, mas com um nome apropriado
```

Evite informações erradas: Exemplo

Imagine encontrar um código assim:

```
int a = 1;
if(O == 1)
   a == O1;
else
   l = 01;
```

Nesse contexto, as variáveis não têm um significado qualquer.

Pode-se confundir o “0” com o “O” e 1 com l (L).

Faça Distinções Significativas

Programadores devem evitar nomear variáveis de maneira arbitrária, com números ou palavras muito comuns.

Repare que no código a seguir o uso das palavras “Info” e “Data” não distingue o sentido das duas Strings.

```
String productInfo; //diferenciando informação
String productData; //diferenciando dados
```

Faça Distinções Significativas

As palavras muito comuns são também redundantes. Imagine encontrar funções como as a seguir ...

```
getActiveAccounts();
getActiveAccountsInfo();
//o erro está em usar as duas juntas,
//podendo confundir na hora da leitura
```

Qual a diferença entre elas?

Faça Distinções Significativas: Exemplo copy chars

Nomes como "a1" e "a2" não fornecem nenhuma informação sobre o que essas variáveis representam ou qual é a sua finalidade no código.

A ausência de nomes significativos dificulta o entendimento, aumentando o trabalho dos desenvolvedores que precisam ler e trabalhar com o código.

```
public void copyChars(char a1[], char a2[]){
    for(int i = 0; i < length; i++){
        a2[i] = a1[i];
    }
}
```

```
public void copyChars(char[] source, char[] destination){
    for (int i = 0; i < source.length; i++){
        destination[i] = source[i];
    }
}
```

Use Nomes Pronunciáveis: Exemplo

É importante nomear códigos e funções com nomes pronunciáveis, afinal se você não conseguir falar o nome ao explicá-lo você irá parecer antiprofissional.

Tente pronunciar o nome do primeiro código a seguir:

```
class  DtaRcrd102{
   private Date genymdhms;
   private Date modymdms;
   private final String pszqint = "102";
}
```

```
class Customer{
   private Date generationTimestamp;
   private Date modificationTimestamp;
   private final String recordId = "102";
}
```

Use nomes passíveis de busca

Nomes de uma só letra ou números não são fáceis de localizar:

```
for (int j=0; j<43; j++){
   s+= (t[j]*4)/5;
}
```

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum=0;
for (int j=0; j< NUMBER_OF_TASKS;j++){
   int realTaskDays = taskEstimate[j] *realDaysPerIdealDay;
   int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
   sum +=realTaskWeeks;
}
```

Note que com esta mudança no código conseguimos localizar de uma forma mais simples e eficiente.

Prefixos de Variáveis Membro

Não há necessidade de se utilizar um prefixo “m_” para indicar uma variável membro (global), pois o próprio editor já indica com cores diferentes para o usuário.

```
public class Part{
    private String m_dsc;
    void setName(String name){
        m_dsc = name;
    }
}
```

```
public class Part{
    String description;
    void setDescription(String description){

        this.description = description;
    }
}
```

Evite o Mapeamento Mental

Mapeamento mental consiste em lembrar e acompanhar informações em um contexto. Quanto mais informações, mais difícil de manter uma visão clara do que está sendo feito.

É importante escrever códigos claros e objetivos, com nomes de variáveis, métodos e funções compreensíveis.

Manter menos informações na cabeça ajuda na compreensão e manutenção do código ao longo do tempo.

Nome de Classes e Métodos

Classes e objetos devem ter nomes com substantivo(s), sempre evitando palavras como *Manager*, *Processor*, *Data* ou *Info*.

```
public class Manager{}
public class Processor{}
//estas classes estão com nomes que já significam algo na Computação.
```

Métodos devem possuir verbos como nomes, para comunicar de maneira clara e concisa o que essa função faz.

```
string name = employee.getName();
customer.setName("mike");
if(paycheck.isPosted())
```

Nomes de Métodos

Quando os construtores estiverem sobrecarregados, adote métodos “factory” estáticos e parâmetros descritivos para aumentar a legibilidade:

```
Complex fulcrumPoint = new Complex(23.0);
```

O segundo código é melhor, pois exprime o que o construtor faz e a semântica do parâmetro.

```
Complex fulcrumPoint = Complex.fromRealNumber(23.0);
```

Não dê uma de Espertinho: Exemplo

Nomes engraçados para variáveis podem ser divertidos, mas podem tornar o código menos profissional e difícil de entender para outras pessoas.

É melhor usar nomes descritivos e claros que possam ser facilmente compreendidos por todos os envolvidos no projeto, para garantir a legibilidade e a manutenção.

```
public static void darOfora(){}
```

```
public static void encerrar(){}
```

Selecione uma Palavra por Conceito

Escolha uma palavra chave para cada conceito. Se você fizer uma função que consulta o backend e ela chamar fetchClients, não crie outra getProducts. Use fetch ou get.

```
public void static getClients(){}
public void static fetchProducts(){}
//Qual a diferença entre fetch e get?
```

Use nomes a partir do Domínio da Solução

São programadores que lerão o código. Logo, está tudo bem em escolher nomes a partir do domínio da solução.

É melhor escolher nomes que os programadores estejam familiarizados do que nomes que apenas os clientes tenham domínio.

Nomes como *AccountVisitor* possuem um significado técnico para um programador ciente do padrão de projeto Visitor.

Use nomes de Domínios do Problema

Quando os programadores tiverem dificuldade em nomear um conceito a partir do domínio da solução, é possível utilizar nomes a partir do vocabulário do cliente.

O programador que fizer manutenção no código pode consultar o cliente em caso de dúvida.

Códigos que têm a ver com conceitos do domínio do problema possuem nomes derivados de tal domínio.

Não Adicione Contextos Desnecessários: Exemplo

Não utilize prefixos para indicar a participação de classes dentro de módulos.

Por exemplo, imagine colocar em várias classes de um aplicativo "*Gas Station Deluxe*" o prefixo GSD. Como ficaria a busca por uma variável com a letra G?

Ajude o IDE a te ajudar, você não vai gostar quando a ferramenta de auto completar lhe oferecer dezenas de resultados a cada vez que você digita a letra G!

Conclusão

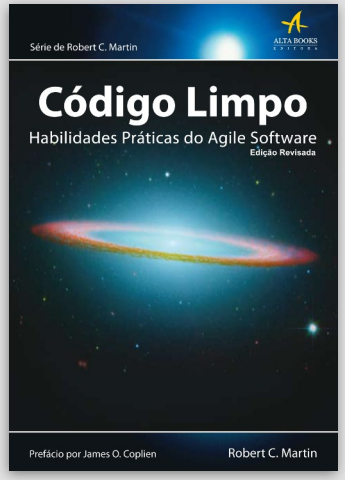
- Escolha nomes que revelem seu propósito
- Código sempre o mais claro possível
- Escolha nomes distinguíveis
- Seja profissional
- Programe pensando que outro profissional irá ver o código
- Evite digitar contextos desnecessários
- O simples em maioria das vezes é a melhor solução

Sobre o autor

Robert C. “Uncle Bob” Martin é desenvolvedor e consultor de software desde 1990. Ele é o fundador e o presidente da Object Mentor, Inc., uma equipe de consultores experientes que orientam seus clientes no mundo todo em C++, Java, C#, Ruby, OO, Padrões de Projeto, UML, Metodologias Agile e eXtreme Programming.

Este conjunto de slides foi elaborado a partir da obra:

MARTIN, Robert. **Código Limpo:** Habilidades Práticas do Agile Software. 1. ed. Rio de Janeiro: Alta Books, 2009.



A equipe

Otávio Lopes, Autor

Aluno e bolsista do PET/ADS
LinkedIn

Lucas Oliveira, Revisor

Professor de Computação, é tutor do PET/ADS desde janeiro de 2023.
LinkedIn