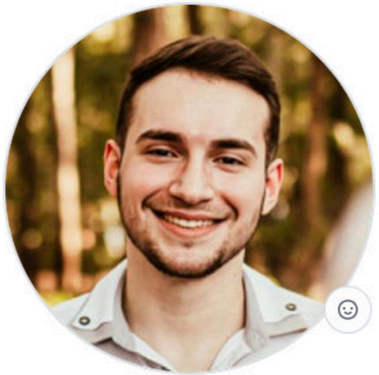
Da Bagunça à Clareza: Clean Code para Avançar na sua Jornada de Desenvolvedor



Matheus Bloise
matheusbloise

Software Engineer at **@pagseguro**

Edit profile

8 followers · **8** following

@pagseguro

Achievements

x2 x3

Organizations

matheusbloise / README.md

Hi there 👋

I'm a software engineer with more than 6 years working in the PHP and JavaScript ecosystem, most of the time working as a backend developer.

My focus as an engineer is on the core concepts, theory and pratice working together, that's becoming me better daily as a software developer and I'm completely passionate to solve real problems with code, this passion took me across several large projects impacting directly thousand people.

"For me quality and good pratices shouldn't have tradeoffs"

Technologies I know and have been working on

🌎 Find me around the web

💼 Connecting and sharing professional updates through LinkedIn.

Pinned

Customize your pins

flunt-php Public
Library for made easy your notifications and validations within domain
PHP 9

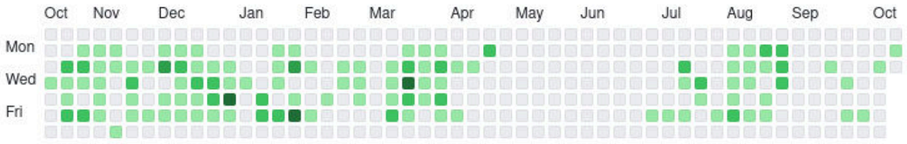
spaceship-clean-architecture-php Public
PHP

Single sign-on to see contributions within the **developer-productivity** organization.

Single sign-on to see contributions within the **ps-corp-platform** organization.

430 contributions in the last year

Contribution settings



Learn how we count contributions

Less More

2024
2023
2022
2021
2020
2019

Introduction to Clean Code

In this presentation, we will explore the **fundamentals** of **clean code**. Writing **maintainable software** is crucial for long-term success. We will discuss key **principles** and **practices** that help developers create code that is not only functional but also easy to read and modify.

Introduction to Clean Code

One of the core principles of clean code is readability. Code should be written in a way that makes it easy for others (and yourself) to understand. Clear naming conventions and consistent formatting are essential to ensure that the code communicates its purpose effectively.

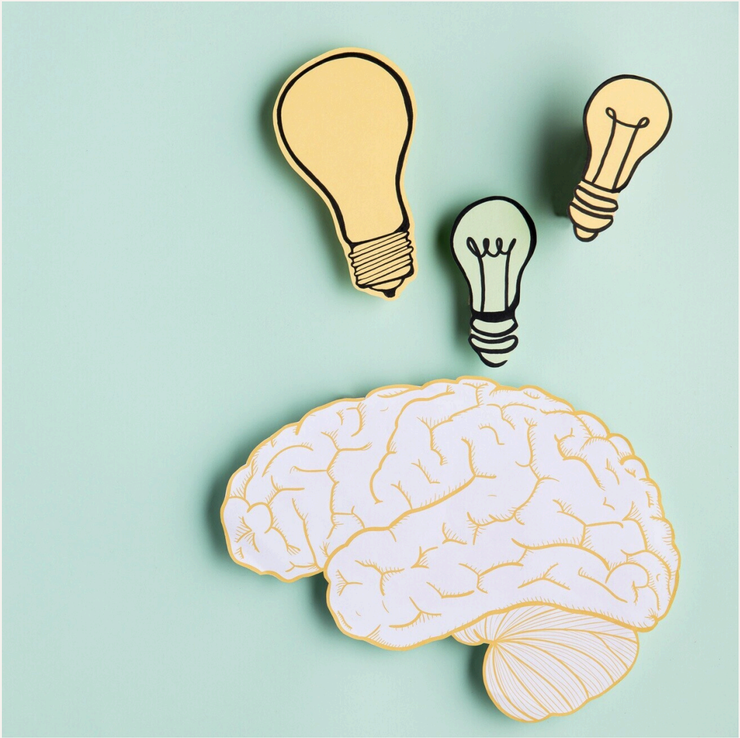
```
2445            if ((lowerChar == Character.ERROR) ||
2446                (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447                if (lowerChar == Character.ERROR) {
2448                    lowerCharArray =
2449                        ConditionalSpecialCasing.toLowerCaseCharArray(this, i, locale);
2450                } else if (srcCount == 2) {
2451            resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCo
2452            continue;
2453                } else {
2454            lowerCharArray = Character.toChars(lowerChar);
2455        }
2456
2457                /* Grow result if needed */
2458                int mapLen = lowerCharArray.length;
2459        if (mapLen > srcCount) {
2460                    char[] result2 = new char[result.length + mapLen - srcCount];
2461                    System.arraycopy(result, 0, result2, 0,
2462                        i + resultOffset);
2463                    result = result2;
2464        }
2465                for (int x=0; x<mapLen; ++x) {
2466                    result[i+resultOffset+x] = lowerCharArray[x];
2467                }
2468                resultOffset += (mapLen - srcCount);
2469            } else {
2470                result[i+resultOffset] = (char)lowerChar;
2471            }
2472        }
2473        return new String(0, count+resultOffset, result);
2474    }
```

```
2445            if ((lowerChar == Character.ERROR) ||
2446                (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447                if (lowerChar == Character.ERROR) {
2448                    lowerCharArray =
2449                        ConditionalSpecialCasing.toLowerCaseCharArray(this, i, locale);
2450                } else if (srcCount == 2) {
2451            resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCo
2452            continue;
2453                } else {
2454            lowerCharArray = Character.toChars(lowerChar);
2455        }
2456
2457                /* Grow result if needed */
2458                int mapLen = lowerCharArray.length;
2459        if (mapLen > srcCount) {
2460                    char[] result2 = new char[result.length + mapLen - srcCount];
2461                    System.arraycopy(result, 0, result2, 0,
2462                        i + resultOffset);
2463                    result = result2;
2464        }
2465                for (int x=0; x<mapLen; ++x) {
2466                    result[i+resultOffset+x] = lowerCharArray[x];
2467                }
2468                resultOffset += (mapLen - srcCount);
2469            } else {
2470                result[i+resultOffset] = (char)lowerChar;
2471            }
2472        }
2473        return new String(0, count+resultOffset, result);
2474    }
```

```
2445            if ((lowerChar == Character.ERROR) ||
2446                (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447                if (lowerChar == Character.ERROR) {
2448                    lowerCharArray =
2449                        ConditionalSpecialCasing.toLowerCaseCharArray(this, i, locale);
2450                } else if (srcCount == 2) {
2451            resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - src
2452            continue;
2453                } else {
2454            lowerCharArray = Character.toChars(lowerChar);
2455        }
2456
2457                /* Grow result if needed */
2458                int mapLen = lowerCharArray.length;
2459        if (mapLen > srcCount) {
2460                    char[] result2 = new char[result.length + mapLen - srcCount];
2461                    System.arraycopy(result, 0, result2, 0,
2462                        i + resultOffset);
2463                    result = result2;
2464        }
2465                for (int x=0; x<mapLen; ++x) {
2466                    result[i+resultOffset+x] = lowerCharArray[x];
2467                }
2468                resultOffset += (mapLen - srcCount);
2469            } else {
2470                result[i+resultOffset] = (char)lowerChar;
2471            }
2472        }
2473        return new String(0, count+resultOffset, result);
2474    }
```

```
2445            if ((lowerChar == Character.ERROR) ||
2446                (lowerChar >= Character.MIN_SUPPLEMENTARY_CODE_POINT)) {
2447                if (lowerChar == Character.ERROR) {
2448                    lowerCharArray =
2449                        ConditionalSpecialCasing.toLowerCaseCharArray(this, i, locale);
2450                } else if (srcCount == 2) {
2451            resultOffset += Character.toChars(lowerChar, result, i + resultOffset) - srcCo
2452            continue;
2453                } else {
2454            lowerCharArray = Character.toChars(lowerChar);
2455        }
2456
2457                /* Grow result if needed */
2458                int mapLen = lowerCharArray.length;
2459        if (mapLen > srcCount) {
2460                    char[] result2 = new char[result.length + mapLen - srcCount];
2461                    System.arraycopy(result, 0, result2, 0,
2462                        i + resultOffset);
2463                    result = result2;
2464        }
2465                for (int x=0; x<mapLen; ++x) {
2466                    result[i+resultOffset+x] = lowerCharArray[x];
2467                }
2468                resultOffset += (mapLen - srcCount);
2469            } else {
2470                result[i+resultOffset] = (char)lowerChar;
2471            }
2472        }
2473        return new String(0, count+resultOffset, result);
2474    }
```

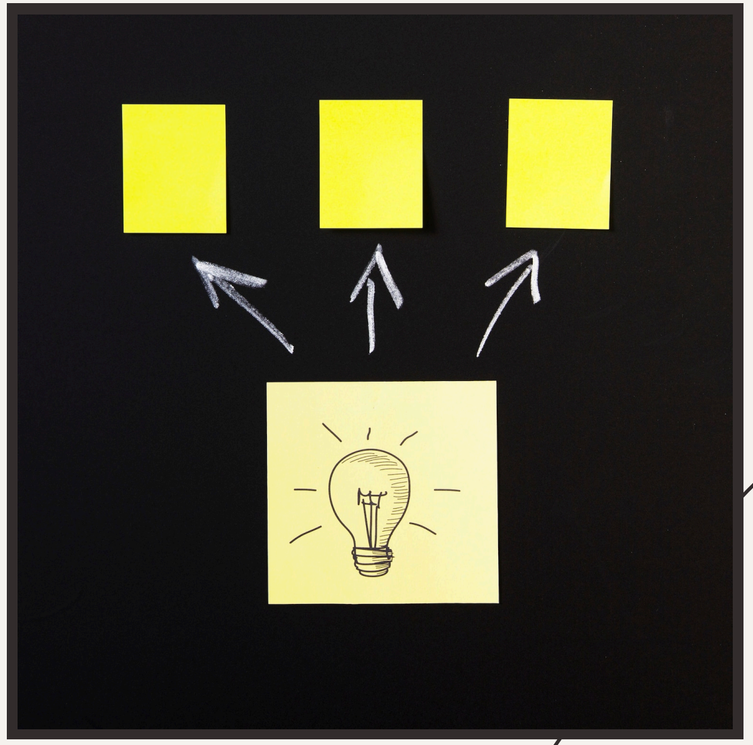

VAI SE CRIANDO
UM CLIMA TERRRÍVEL



Bjarne Stroustrup: "The logic should be straightforward so that it is difficult to hide defects."

KISS Principle

The **KISS** (Keep It Simple, Stupid) principle advocates for simplicity in design and implementation. Complex solutions can lead to **bugs** and **maintenance headaches**. Strive for simplicity by using straightforward approaches and avoiding unnecessary complexity in your code.



DRY Principle

The **DRY** (Don't Repeat Yourself) principle emphasizes the importance of **reducing duplication** in code. By avoiding redundancy, you make your codebase easier to maintain, as changes need to be made in only one place. This principle promotes **modularity** and **reusability**.

Testing and Refactoring

Regular **testing** and **refactoring** are vital for maintaining clean code. Automated tests help ensure that your code behaves as expected, while refactoring allows you to improve the structure without changing functionality. Together, they contribute to a **robust** and **flexible codebase**.

Thanks!