

Descriptive Variable Names ✨


```
// Good Practice  
double temperatureCelsius = 20.5;
```

When you use descriptive variable names, it's immediately clear that `temperatureCelsius` represents a temperature value in Celsius.

In contrast, using `temp` doesn't provide enough context, making the code less understandable.




Modular Functions



```
// Good Practice
public double calculateInterest(double principal, double rate, int years) {
    // Calculate interest here
}
```

In a banking application, if you need to calculate interest for different scenarios, modular functions like `calculateInterest` allow you to reuse the logic and make changes without affecting the rest of the code.

In contrast, a single, lengthy function (`processBanking`) becomes difficult to understand and maintain.



Comments for

Clarity

```
// Good Practice
// Calculate interest based on principal, rate, and time
public double calculateInterest(double principal, double rate, int years) {
    // Calculate interest here
}
```

When you're working on a team project, well-placed comments provide a clear explanation of what the function does.

This helps your team members understand and collaborate effectively. Without comments, the code may be confusing, leading to misunderstandings and potential errors.



Proper Error Handling



```
// Good Practice
try {
    // Code that may throw exceptions
} catch (Exception e) {
    // Handle the exception gracefully
}
```

Consider a user registration process. Proper error handling ensures that if something goes wrong, the application doesn't crash.

Users receive meaningful error messages, and you can identify and address issues quickly.

Without error handling, the application might crash or provide vague error messages, leading to a poor user experience.



Version Control ✨ and Branching

```
# Good Practice (using Git)
git init
git checkout -b feature/cool-new-feature
git add .
git commit -m "Implement cool new feature"
git checkout main
git merge feature/cool-new-feature
```

When working on a collaborative project, using version control (e.g., Git) with branching keeps the codebase organized.

You can work on new features without affecting the main code. It also provides a safety net to roll back changes if something goes wrong.



Magic Numbers ✨

```
// Good Practice  
final int SECONDS_IN_AN_HOUR = 3600;  
int hours = seconds / SECONDS_IN_AN_HOUR;
```

When you encounter the term "3600" in your code, it's not immediately clear what it represents.

By using named constants like SECONDS_IN_AN_HOUR, you make the code more understandable and maintainable.



Small Functions ✨

```
// Good Practice  
public void processPayment() {  
    validatePayment();  
    calculateTotal();  
    sendPayment();  
}
```

In an e-commerce checkout process, breaking down the payment process into smaller, modular functions (as in the good practice example) makes the code easier to read, understand, and maintain.

In contrast, a long, complex function (bad practice) is difficult to manage.



Inconsistent Formatting



```
// Good Practice (consistent indentation)
if (condition) {
    // Code block
}
```

In a team project, following a consistent coding style and formatting (as shown in the good practice example) is crucial.

It ensures that team members can read and understand the code without confusion.

Inconsistent formatting can lead to misunderstandings and errors.



Local Variables ✨

```
// Good Practice  
public void processOrder(Order order) {  
    // Process the order  
}
```

Keeping variables localized (as in the good practice example) ensures that changes to a variable only affect the scope of the function, reducing the risk of unexpected side effects.

Using global variables can lead to bugs that are challenging to trace.

