Clean Code Development Cheat Sheet

Meaningful Names

- **Use descriptive and meaningful names for variables, functions, and classes.**
- **Avoid single-letter names or ambiguous abbreviations.**

Comments

- **Write self-explanatory code, minimize the need for comments.**
- **When necessary, use comments to clarify complex logic or non-obvious decisions.**

Functions

- **Keep functions small and focused on a single responsibility (Single Responsibility Principle).**
- **Aim for functions to be ideally no longer than 20 lines.**
- **Use meaningful names for functions that convey their purpose.**

Formatting

- **Follow a consistent indentation style (e.g., tabs or spaces).**
- **Use consistent and clear formatting throughout the codebase.**

Error Handling

- **Use exceptions for exceptional situations, not for control flow.**
- **Handle errors gracefully and provide informative error messages.**

STL Usage

- **Leverage the C++ Standard Library (STL) for common data structures and algorithms.**
- **Be familiar with container classes (e.g., ‘std::vector’, ‘std::map’) and algorithms.**

Class Design

- **Follow the Single Responsibility Principle (SRP) for classes.**
- **Prefer composition over inheritance.**
- **Use access specifiers (public, private, protected) judiciously.**

Avoid Magic Numbers

- **Replace numeric constants with named constants or enums to improve code readability.**

Documentation

- **Provide clear and concise documentation for public interfaces.**
- **Document the purpose of classes, functions, and significant code sections.**

Version Control

- **Use version control (e.g., Git) effectively.**
- **Make frequent, small commits with clear commit messages.**

Testing

- **Write unit tests for critical functionality.**
- **Follow the Arrange-Act-Assert (AAA) pattern in your test cases.**

Const-Correctness

- **Use ‘const’ wherever possible to indicate immutability.**
- **Const-correctness helps prevent unintended modifications and enhances code readability.**