rules.sonarsource.com

C++ static code analysis: Classes should not contain both public and private data members

2 minutes

Mixing (non-const) `public` and `private` data members is a bad practice because it causes confusion about the intention of the class:

- If the class is a collection of loosely related values, all the data members should be `public`.
- On the other hand, if the class is trying to maintain an invariant, all the data members should be `private`.

  If we mix data members with different levels of accessibility, the purpose of the class is muddled.

Noncompliant Code Example

```
class MyClass // Noncompliant
{
public:
  int firstNumber1() const { return firstNumber; }
  void setFirstNumber(int firstNumber) { this->firstNumber =
firstNumber; }
  int secondNumber = 2;
  const int constNumber = 0; // const data members are fine
private:
  int firstNumber = 1;
};
```

Compliant Solution

```
class MyClass // Depending on the case, the solution might be
different. Here, since this class does not enforce any invariant, we
make all the data members public
{
public:
  int firstNumber;
  int secondNumber;
  const int constNumber = 0;
};
```

Exceptions

Since `const` data members cannot be modified, it's not breaking encapsulation to make a const value public, even in a class that enforces an invariant.

See

- C++ Core Guidelines C.134: Ensure all non-const data members have the same access level
- C++ Core Guidelines C.9: Minimize exposure of members