rules.sonarsource.com

C++ static code analysis: Constructors and destructors should only use defined methods and fields

3 minutes

Calling methods or fields which are not initialized in constructors or destructors can lead to undefined behavior.

For example:

Calling an overridable member function from a constructor or destructor could result in unexpected behavior when instantiating a subclass which overrides the member function.

- By contract, the subclass class constructor starts by calling the parent class constructor.
- The parent class constructor calls the parent member function and not the one overridden in the child class, which is confusing for child class' developer.
- It can produce an undefined behavior if the member function is pure `virtual` in the parent class.

Noncompliant Code Example

```
class Parent {
```

```
  public:
    Parent() {
      method1();
      method2(); // Noncompliant; confusing because
Parent::method2() will always been called even if the
method is overridden
    }
    Parent(int i):field(i) {}
    virtual ~Parent() {
      method3(); // Noncompliant; undefined behavior (ex:
throws a "pure virtual method called" exception)
    }
  protected:
    int field;

    int          method1() { /*...*/ }
    virtual void method2() { /*...*/ }
    virtual void method3() = 0; // pure virtual
};

class Child : public Parent {
  public:
    Child() { // leads to a call to Parent::method2(), not
Child::method2()
    }
    Child() : Parent(field) {} // Noncompliant; "field" is not
initialized yet
    Child() : Parent(method1()) {} // Noncompliant;
"method1" is not initialized yet
    virtual ~Child() {
```

```
      method3(); // Noncompliant; Child::method3() will
always be called even if a child class overrides method3
    }
  protected:
    void method2() override { /*...*/ }
    void method3() override { /*...*/ }
};
```

Compliant Solution

```
class Parent {
  public:
    Parent() {
      method1();
      Parent::method2(); // acceptable but poor design
    }
    virtual ~Parent() {
      // call to pure virtual function removed
    }
  protected:
    void         method1() { /*...*/ }
    virtual void method2() { /*...*/ }
    virtual void method3() = 0;
};

class Child : public Parent {
  public:
    Child() {
    }
    virtual ~Child() {
```

```
    method3(); // method3() is now final so this is okay
  }
 protected:
  void method2() override { /*...*/ }
  void method3() final    { /*...*/ } // this virtual function is
"final"
};
```

See

- CERT, MET05-J. - Ensure that constructors do not call overridable methods
- CERT, OOP50-CPP. - Do not invoke virtual functions from constructors or destructors