rules.sonarsource.com

C++ static code analysis: Objects should not be sliced

2 minutes

Slicing is what happens when an object from a derived type is cast to an object of one of its base classes. When this happens, the newly created object will not have any of the member variables that are specific to the derived type. This is usually not what was intended. Most of the time, polymorphism is achieved by casting a reference or a pointer to an object of the derived class to a reference or a pointer to an object of a base class.

Note that it’s usually a good idea to design a base class so that slicing cannot happen (it can be *abstract*, or *non-copiable*).

Noncompliant Code Example

```
struct Shape {
  Point position;
  virtual ~Shape() = default;
};
struct Circle : public Shape {
  double radius;
};

void f() {
  vector<Shape> myShapes;
  Circle c;
  myShapes.push_back(c); // Noncompliant. What will be
in the vector is a Shape, not a Circle, and won't contain
any radius
}
```

Compliant Solution

```
void f() {
  vector<unique_ptr<Shape>> myShapes;
  auto c = make_unique<Circle>();
  myShapes.push_back(move(c)); // Compliant. A pointer
to Shape that happens to point to a Circle
}
```

See

- C++ Core Guidelines ES.63 - Don’t slice