**Open/Closed Principle (OCP):** Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. In other words, the behavior of a module can be extended without modifying its source code.

This encourages the use of interfaces and abstract classes to allow for easy extension without altering existing code.

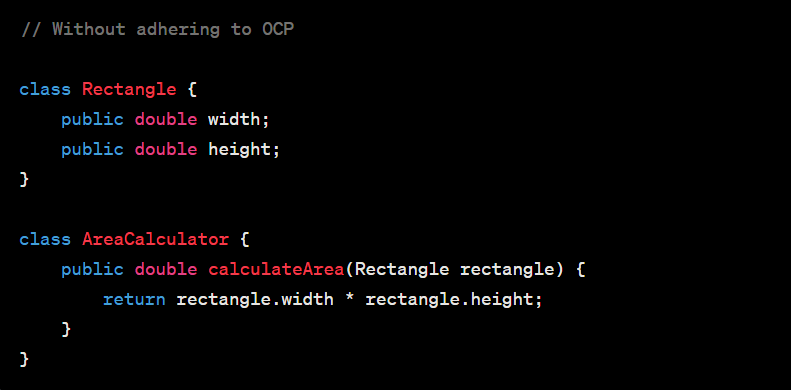
SRP is the prerequisite for OCP.

**Key points of the Open/Closed Principle:**

1. **Open for Extension:** The design should allow adding new functionality or behavior without altering existing code. This is typically achieved through the use of interfaces, abstract classes, and polymorphism.
2. **Closed for Modification:** Once a module (class, function, etc.) is implemented and working correctly, its source code should not be modified to add new features or behaviors. Instead, new features should be added through extensions or modifications that do not affect the existing code.

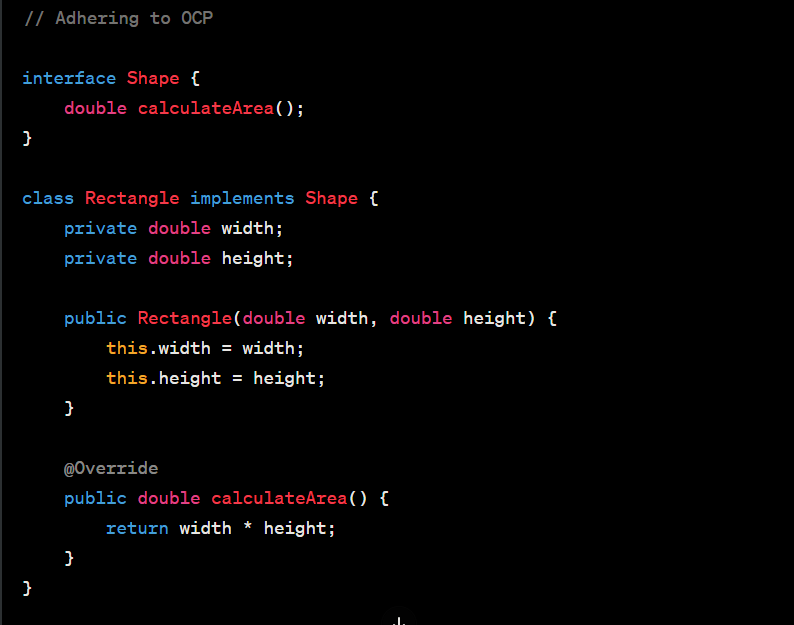
By adhering to the Open/Closed Principle, software systems become more modular and less prone to introducing bugs when new features are added. This principle promotes the use of abstractions and encourages the creation of well-defined interfaces that can be extended by subclasses or implementations.

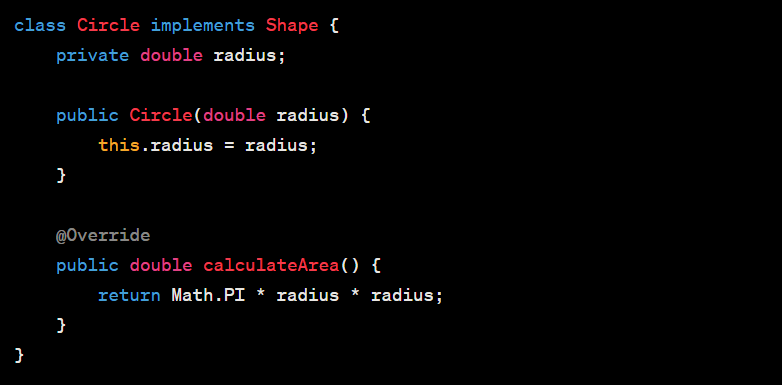
Here's a simple example in Java to illustrate the Open/Closed Principle:



In the above example, if you want to extend the system to support other shapes, you might end up modifying the `AreaCalculator` class.

Now, let's modify it to adhere to OCP:





In this refactored example, the `AreaCalculator` class has been replaced with an `interface Shape`, and specific shapes like `Rectangle` and` Circle` implement this interface. New shapes can be added by creating new classes that implement the `Shape` interface, without modifying the existing code. This adheres to the Open/Closed Principle.