OPEN/CLOSED PRINCIPLE

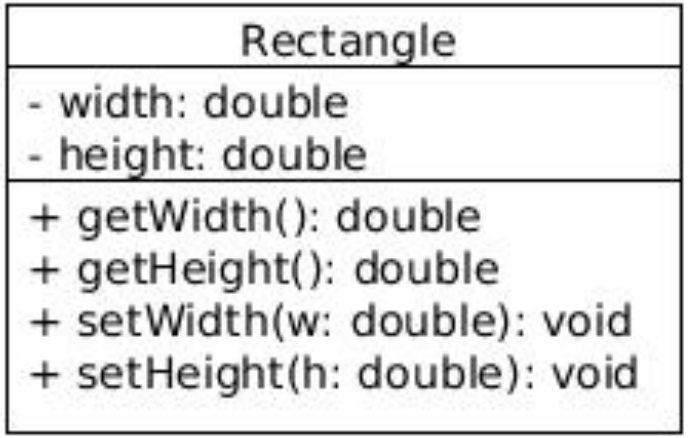
OCP

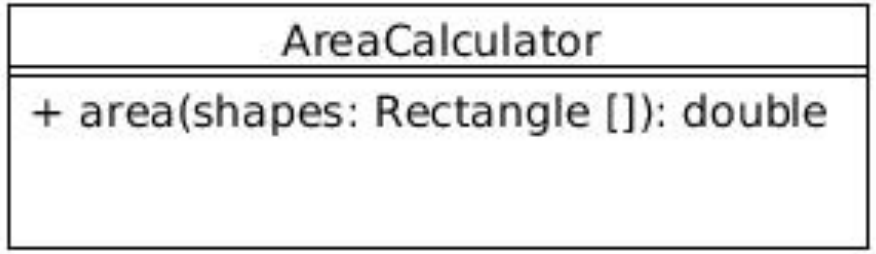
- Software entities (classes, modules, functions, etc.) should be **open for extension, but closed for modification**.
- Add new features not by modifying the original class, but rather by extending it and adding new behaviours, or by adding plugin capabilities.
- “I’ve heard it said that the OCP is wrong, unworkable, impractical, and not for real programmers with real work to do. The rise of plugin architectures makes it plain that these views are utter nonsense. On the contrary, a strong plugin architecture is likely to be the most important aspect of future software systems.” [Uncle Bob, The Open Closed Principle]

OPEN/CLOSED PRINCIPLE

OCP

- An example using inheritance
- The `area` method calculates the area of all Rectangles in the given array.
- What if we need to add more shapes?

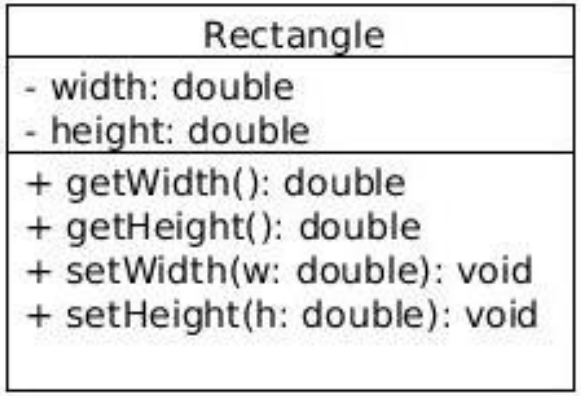


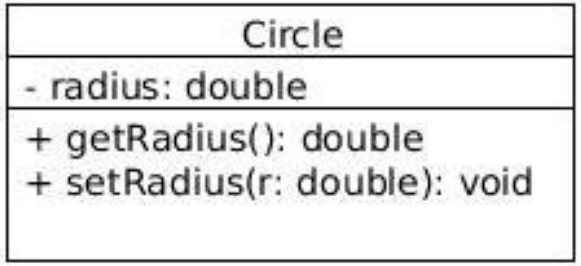


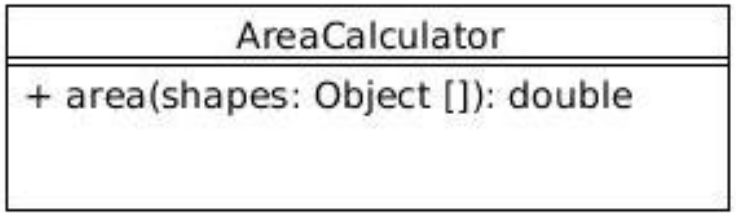
OPEN/CLOSED PRINCIPLE

OCP

- We might make it work for circles too.
- We could implement a `Circle` class and **rewrite** the `area` method to take in an **array of Objects** (using `isinstanceof` to determine if each `Object` is a `Rectangle` or a `Circle` so it can be cast appropriately).







- But what if we need to add even more shapes?

OPEN/CLOSED PRINCIPLE

OCP

- With this design, we can add any number of shapes (open for extension) and we don't need to re-write the `AreaCalculator` class (closed for modification).