Underpins other principles

* All member variables should be private; that ensures that dependents are closed to changes in the member variables.
* Avoid global variables; that ensures code that accesses the global is closed to abuse of the global.

Apply abstraction to parts of the program that are likely to change.

When requirements change, extend the behavior of modules, by adding new code, NOT by changing code that already exists.

Paradox

* Open for extension: make the module behave in new ways, as the module requirements change.
* Close for modification: but \*\*without\*\* changing the source code.

How? Use abstractions

* If a module depend on an abstraction (closure is based on abstraction)
* Then we can extend the module without changing it
* By creating new derivatives of the abstraction

Benefits:

* We can make sure the existing code does not break.

Sometimes open/closed may avoid change in module but require a change in the dependency injection container.

Closed against

* A draw method ought to be closed against new types of shapes
* It could also be closed against a new order in which to draw the shapes.
* A module cannot be closed against \*all\* changes.
* Ergo, only close modules to likely changes.

Methods of closure

* Generally requires the use of an abstraction.
* I.e. an abstraction for Shape or an abstraction for ordering.