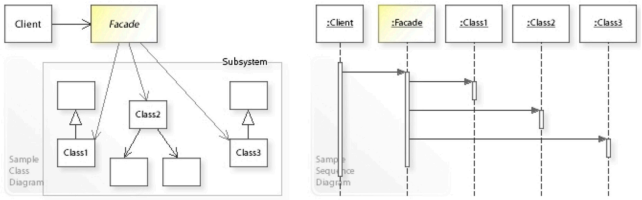
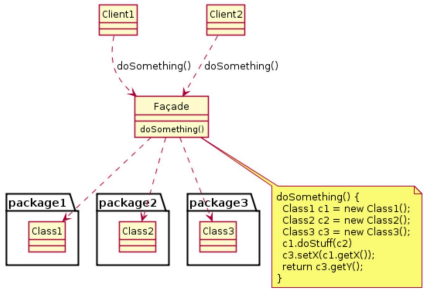
① Design pattern description

Facade c (fachada/portada) — Structural Pattern



UML class diagram



Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

The **facade pattern** (also spelled *façade*) is a software-design pattern commonly used with object-oriented programming. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code. A facade can:

- improve the readability and usability of a software library by masking interaction with more complex components behind a single (and often simplified) API
- provide a context-specific interface to more generic functionality (complete with context-specific input validation)
- serve as a launching point for a broader refactor of monolithic or tightly-coupled systems in favor of more loosely-coupled code

What problems can the Facade design pattern solve? [2]

- To make a complex subsystem easier to use, a simple interface should be provided for a set of interfaces in the subsystem.
- The dependencies on a subsystem should be minimized.

Clients that access a complex subsystem directly refer to (depend on) many different objects having different interfaces (tight coupling), which makes the clients hard to implement, change, test, and reuse.

What solution does the Facade design pattern describe?

Define a `Facade` object that

- implements a simple interface in terms of (by delegating to) the interfaces in the subsystem and
- may perform additional functionality before/after forwarding a request.

This enables to work through a `Facade` object to minimize the dependencies on a subsystem.

Usage

A Facade is used when an easier or simpler interface to an underlying object is desired.[3] Alternatively, an adapter can be used when the wrapper must respect a particular interface and must support polymorphic behavior. A decorator makes it possible to add or alter behavior of an interface at run-time.

Pattern	Intent
Adapter	Converts one interface to another so that it matches what the client is expecting
Decorator	Dynamically adds responsibility to the interface by wrapping the original code
Facade	Provides a simplified interface

not necessary to add functionalities, just make it easy to use.

The facade pattern is typically used when

- a simple interface is required to access a complex system,
- a system is very complex or difficult to understand,
- an entry point is needed to each level of layered software, or
- the abstractions and implementations of a subsystem are tightly coupled.

(2) Design pattern example

```python
# Complex computer parts
class CPU(object):
    """
    Simple CPU representation.
    """
    def freeze(self):
        print("Freezing processor.")

    def jump(self, position):
        print("Jumping to:", position)

    def execute(self):
        print("Executing.")
```

```python
class Memory(object):
    """
    Simple memory representation.
    """
    def load(self, position, data):
        print("Loading from {0} data: '{1}'.".format(position, data))
```

```python
class SolidStateDrive(object):
    """
    Simple solid state drive representation.
    """
    def read(self, lba, size):
        return "Some data from sector {0} with size {1}".format(lba, size)
```

```python
class ComputerFacade(object):
    """
    Represents a facade for various computer parts.
    """
    def __init__(self):
        self.cpu = CPU()
        self.memory = Memory()
        self.ssd = SolidStateDrive()

    def start(self):
        self.cpu.freeze()
        self.memory.load("0x00", self.ssd.read("100", "1024"))
        self.cpu.jump("0x00")
        self.cpu.execute()
```

```python
computer_facade = ComputerFacade()
computer_facade.start()
```

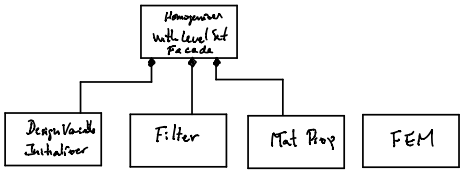
Output:

```
Freezing processor.
Loading from 0x00 data: 'Some data from sector 100 with size 1024'.
Jumping to: 0x00
Executing.
```

③ Existing Example in FEM-MAT-OO ?

Not explicitly

④ Design pattern proposal in FEM-MAT-OO ?



HomogenizerWithLevelSetFacade

Ψ = DesignVariableInitializer.createLevelSet()

ρ = Filter.filter(Ψ)

C = MatPropertiesGlobalConstitutiveTensor(ρ)

F = FEM.create(Settings) (macro)

F.setMatProp(C)

F.solve()