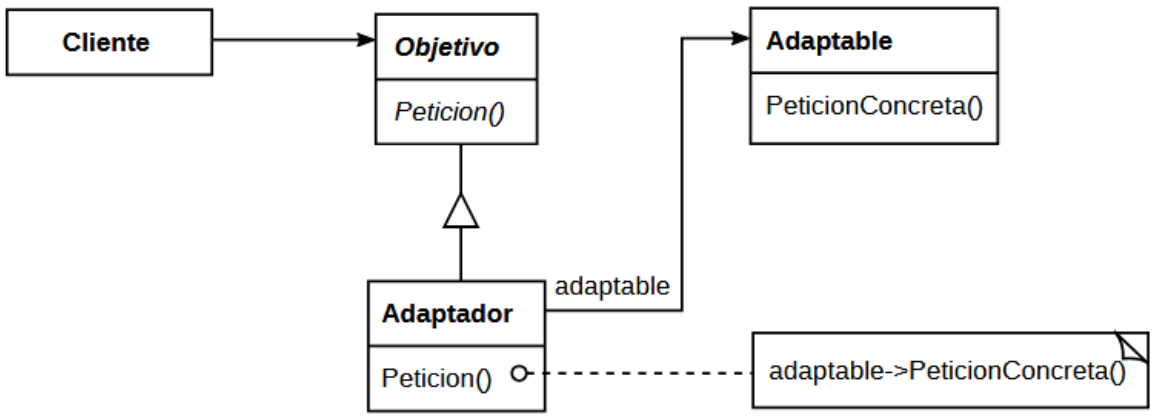
Patrones

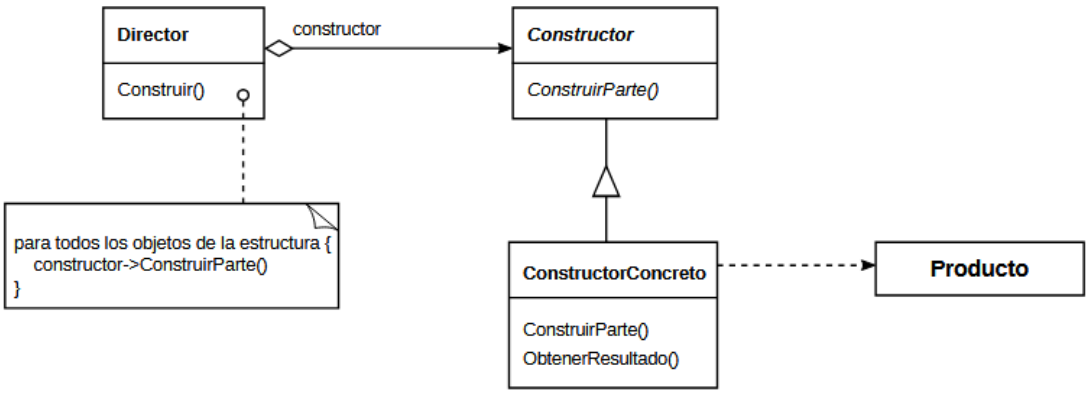
Adapter

- **Propósito** → *Adaptar la intefaz de una clase en otra para poder utilizarla*
- **Participantes** → Cliente, Objetivo, Adaptador, Adaptable
  - El **Objetivo** define la interfaz específica del dominio que usa el Cliente
  - El **Cliente** colabora con objetos que se ajustan a la interfaz Objetivo
  - El **Adaptable** define una interfaz existente que necesita ser adaptada
  - El **Adaptador** adapta la interfaz de Adaptable a la interfaz Objetivo



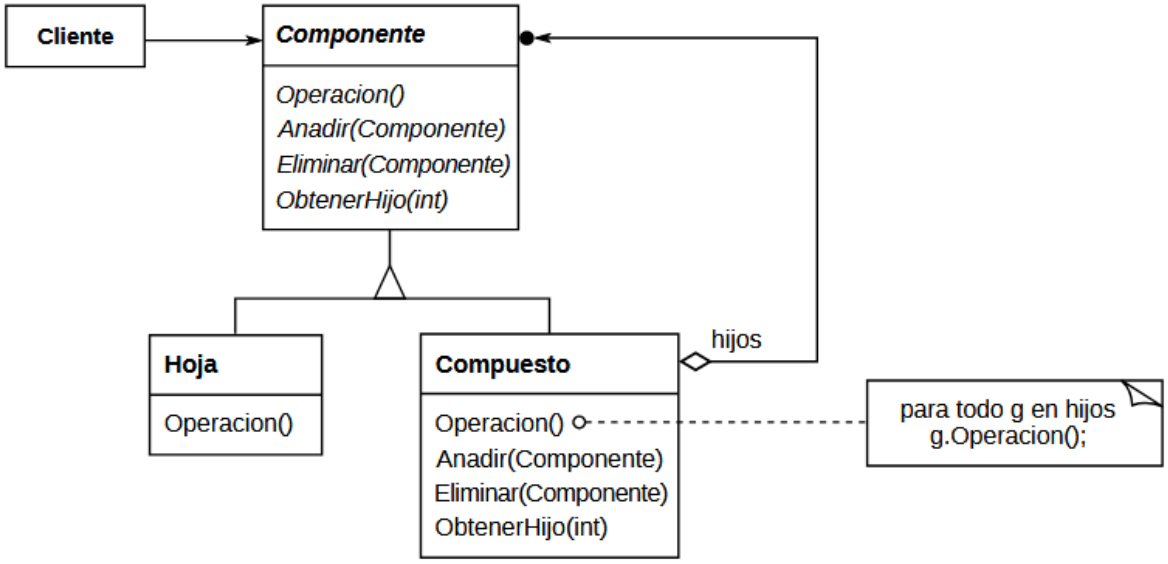
Builder

- **Propósito** → *Separar la construcción de la representación de un objeto*
- **Participantes** → Director, Constructor, Constructor Concreto, Producto
- **Aplicabilidad**
  - El algoritmo de creación debiera ser independiente de las partes de que se compone el objeto, y de como se ensamblan.
  - El proceso de construcción debe permitir diferentes representaciones del objeto que está siendo construido



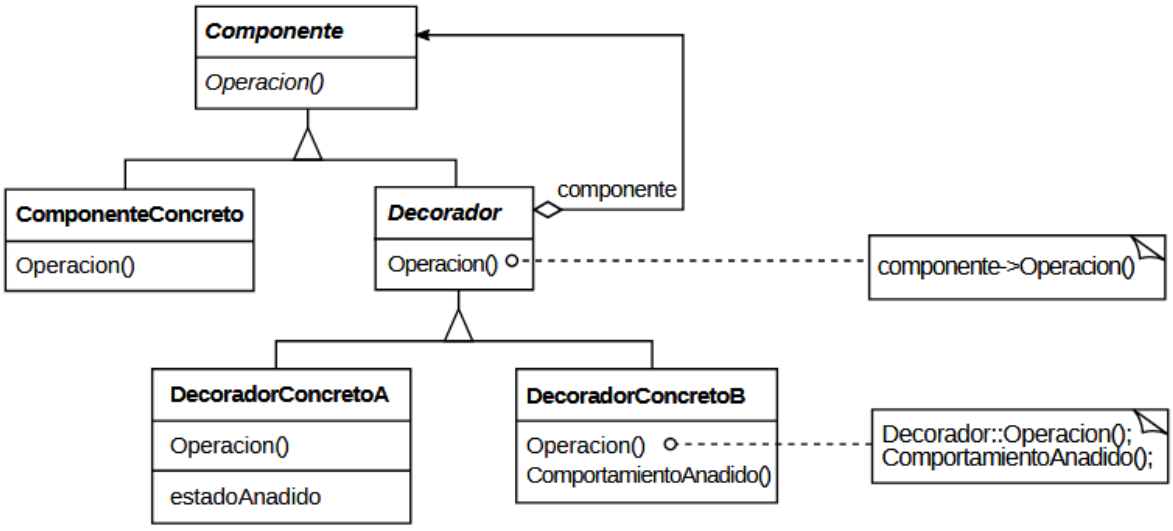
Composite

- **Propósito** → *Crear objetos compuestos y permitir que estos y los simples se traten igual*
- **Participantes** → Cliente, Componente, Hoja, Compuesto
- **Implementación** → Balancear la transparencia y la seguridad, y ver cual se prefiere
  - Si se opta por mantener la transparencia, los métodos que agregue el Compuesto también deberán estar en Componente (como métodos abstractos quizas)
  - Si se opta por mantener la seguridad, las interfaces de los Compuestos y de los Componentes se diferenciarán, pero no se correrá el riesgo de ejecutar una operación Añadir(Componente) en una Hoja



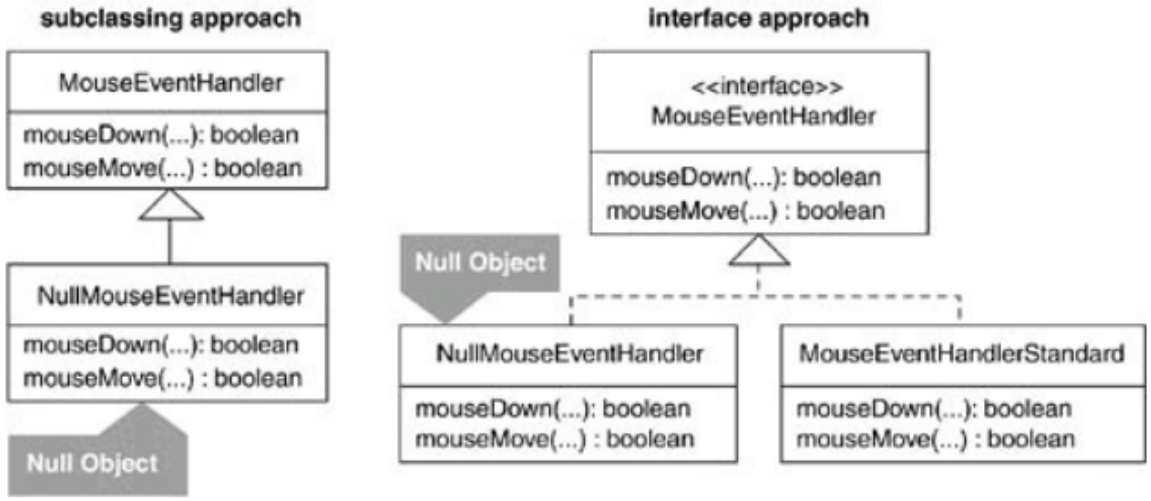
Decorator

- **Propósito** → *Asignar responsabilidades adicionales de manera dinámica a un objeto*
- **Participantes** → Componente, Componente Concreto, Decorador, Decorador Concreto



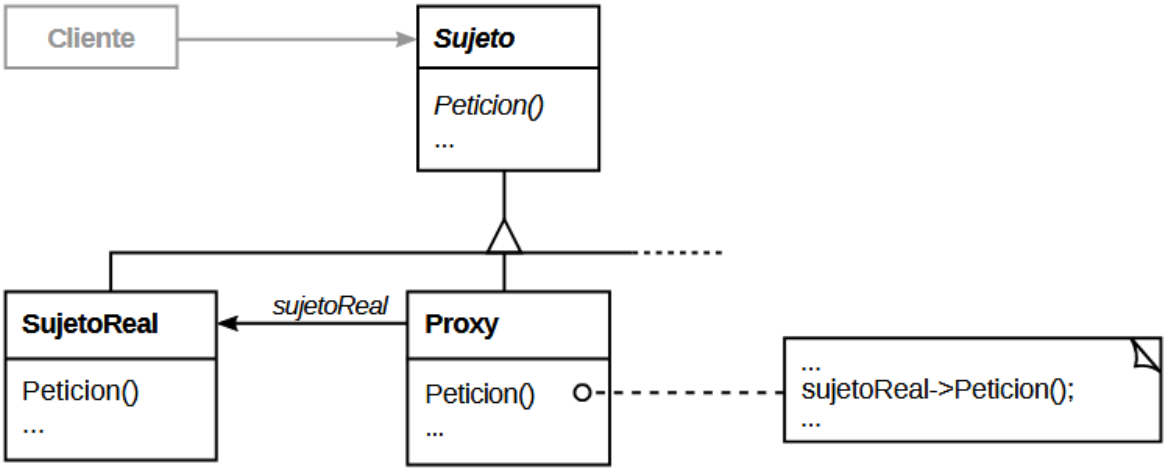
Introduce Null Object

- **Propósito** → *Quitar toda la lógica que controla los nulls en un objeto o jerarquía de clases*
- **Participantes** → Clase, Clase Nula
- **Implementación** → Puede implementarse por herencia o por interfaz



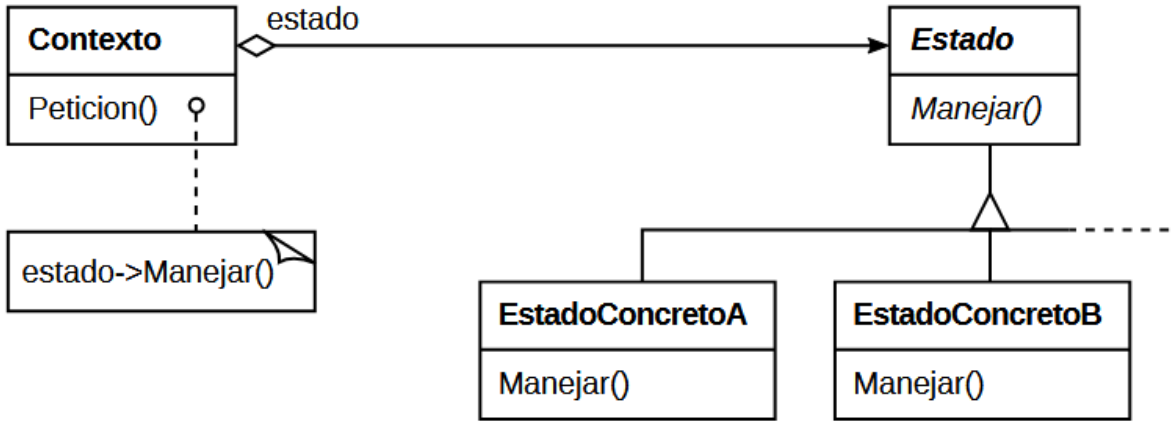
Proxy

- **Propósito** → Controla el acceso a un objeto por medio de un representante
- **Participantes** → Sujeto, Sujeto Real, Proxy



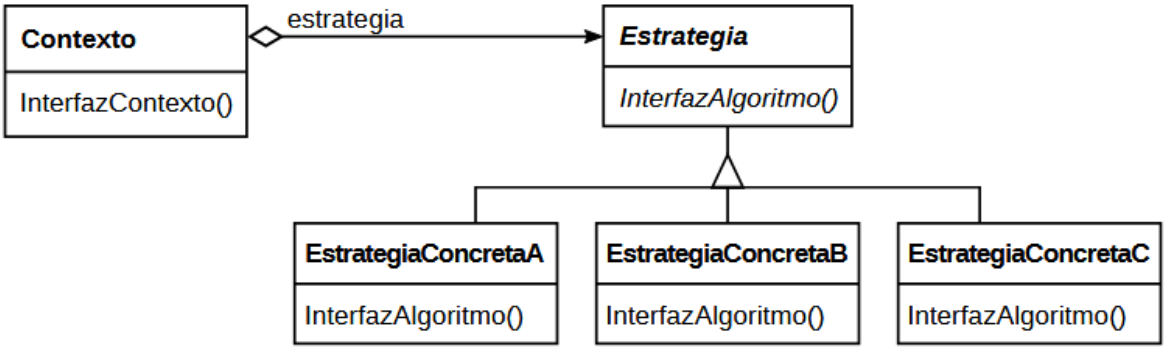
State

- **Propósito** → *Permitir modificar el comportamiento cada vez que cambie el estado interno*
- **Participantes** → Contexto, Estado, Estado Concreto
- **Implementación** → Requerirá pasar el contexto como parámetro al objeto Estado



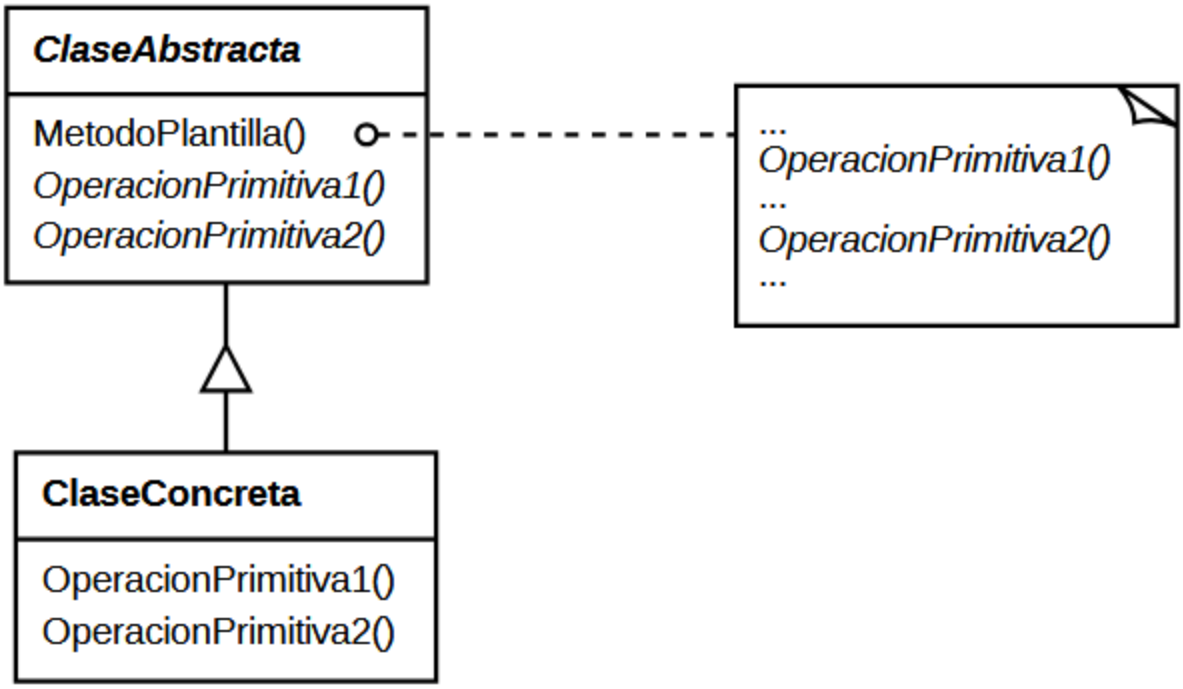
Strategy

- **Propósito** → *Permite que un algoritmo varíe independientemente de los clientes que lo usan*
- **Participantes** → Contexto, Estrategia, Estrategia Concreta
- **Implementación** → Un método puede requerir que se pasen varios datos que tiene el Contexto, y quizás resulte mejor pasarle el objeto Contexto mismo o no, hay que balancear seguridad y eficiencia



Template Method

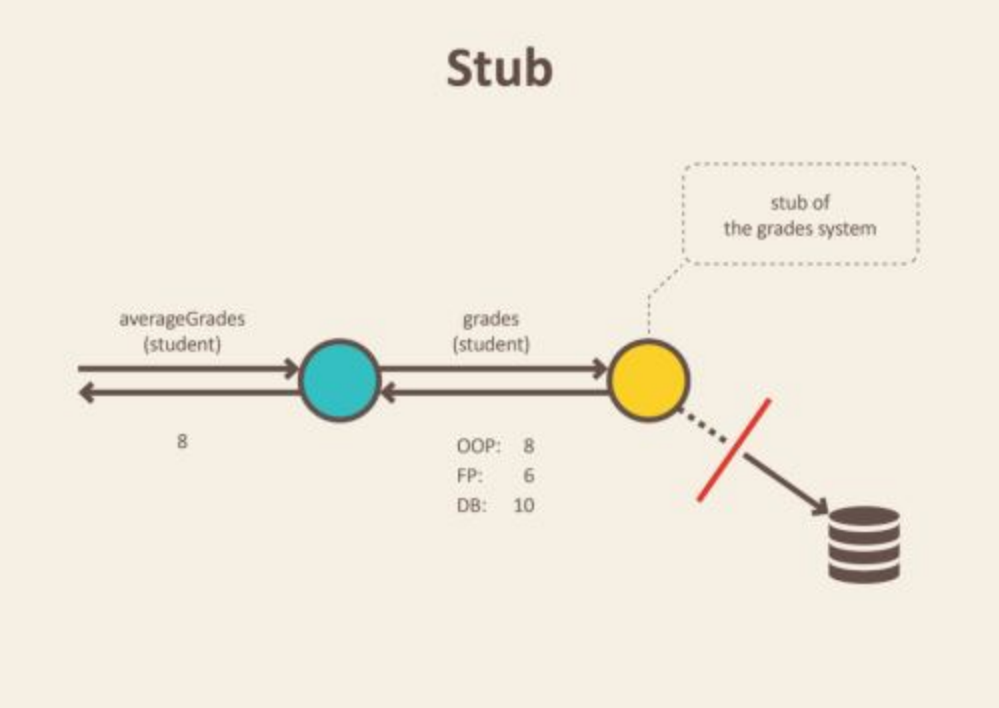
- **Propósito** → *Definir el esqueleto de un método con partes implementables por subclases*
- **Participantes** → Clase Abstracta, Clase Concreta



Test Doubles

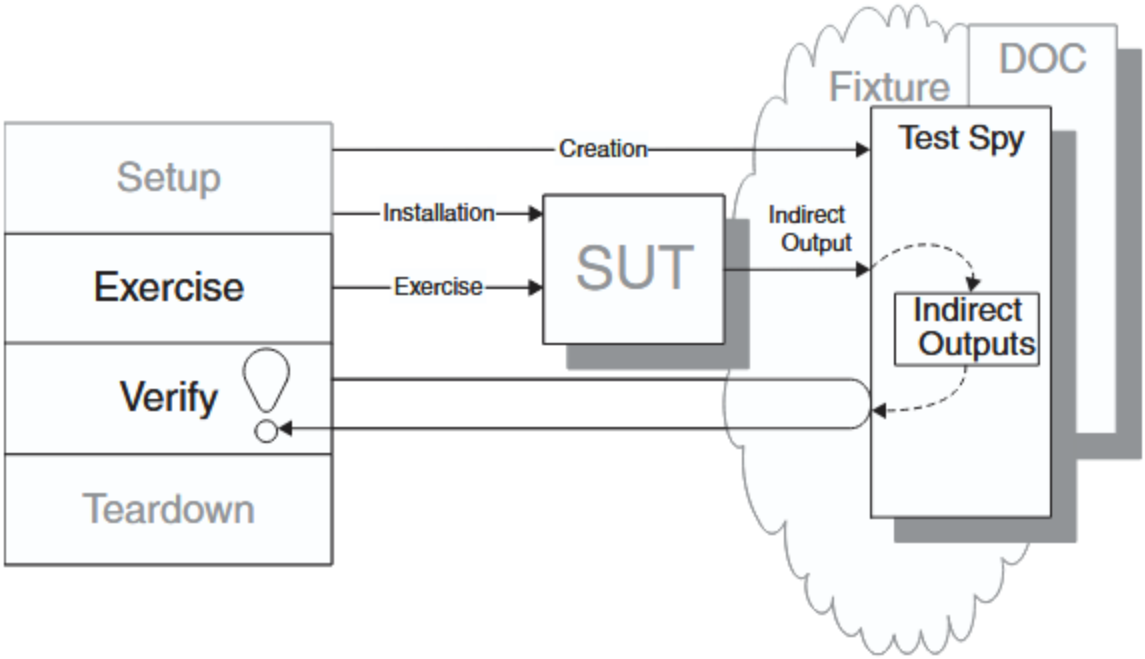
Test Stub

- **Propósito** → *Reemplazar un objeto real por un objeto específico para el test, que meta los inputs indirectos deseados al sistema bajo testing*



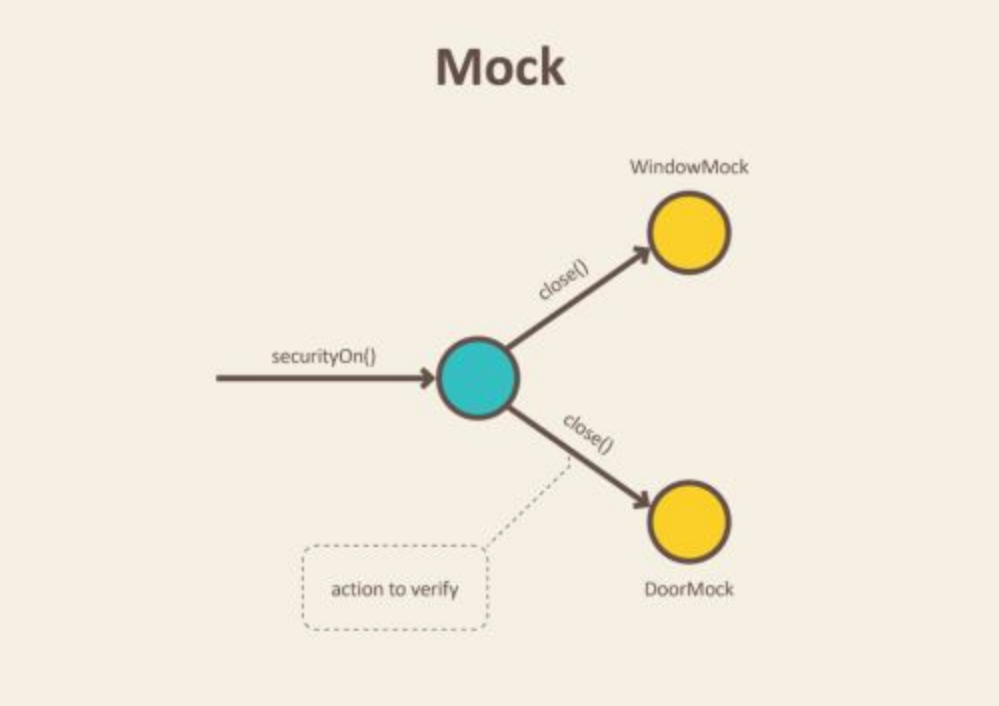
Test Spy

- **Propósito** → *Captura los outputs ndirectos hechos hacia otro componente del sistema para permitir su verificación. Es como un punto de observación*



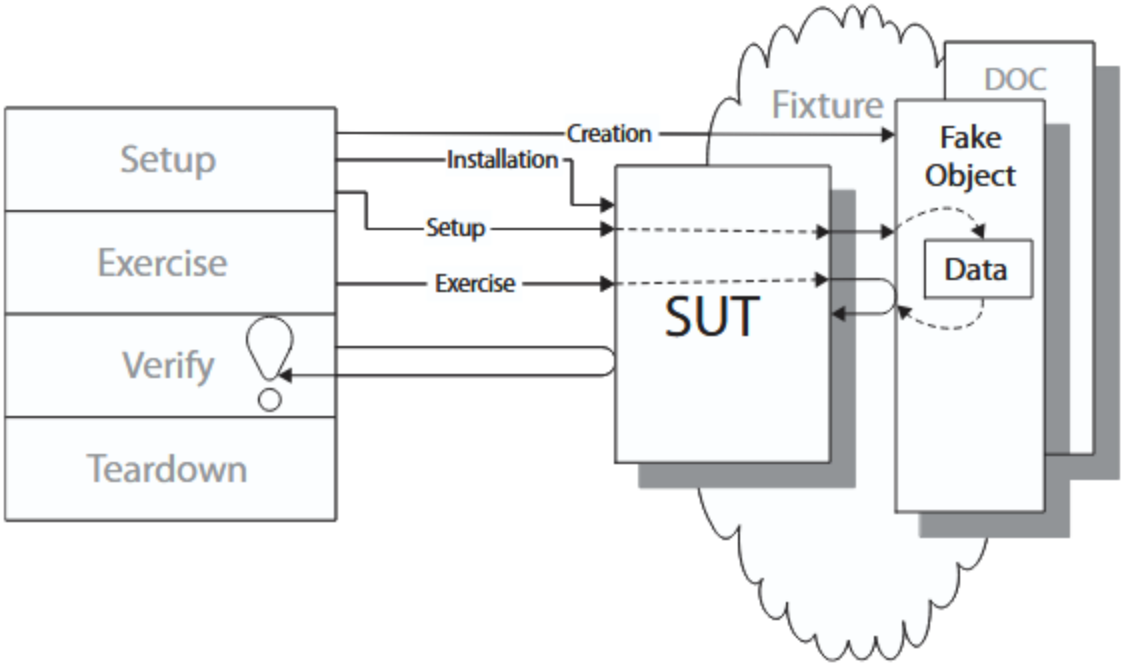
Mock Object

- **Propósito** → *Utilizar cuando se quiera verificar lógica de manera independiente, sin preocuparse por los inputs que se tengan que recibir de otros componentes de Software.*
- En varias situaciones, el contexto en el que opera el sistema influencia mucho el comportamiento de este. En otros casos, debemos mirar "adentro" del sistema para determinar si el comportamiento esperado ocurrió.



Fake Object

- **Propósito** → *Reemplazamos un componente del cual el sistema depende por un objeto falso mucho mas simple.*


Setup
Exercise
Verify
Teardown
Creation
Installation
Setup
Exercise
Fixture
SUT
DOC
Fake
Object
Data