Patrones de Diseño de Software

PATRONES COMPORTAMIENTO - STRATEGY

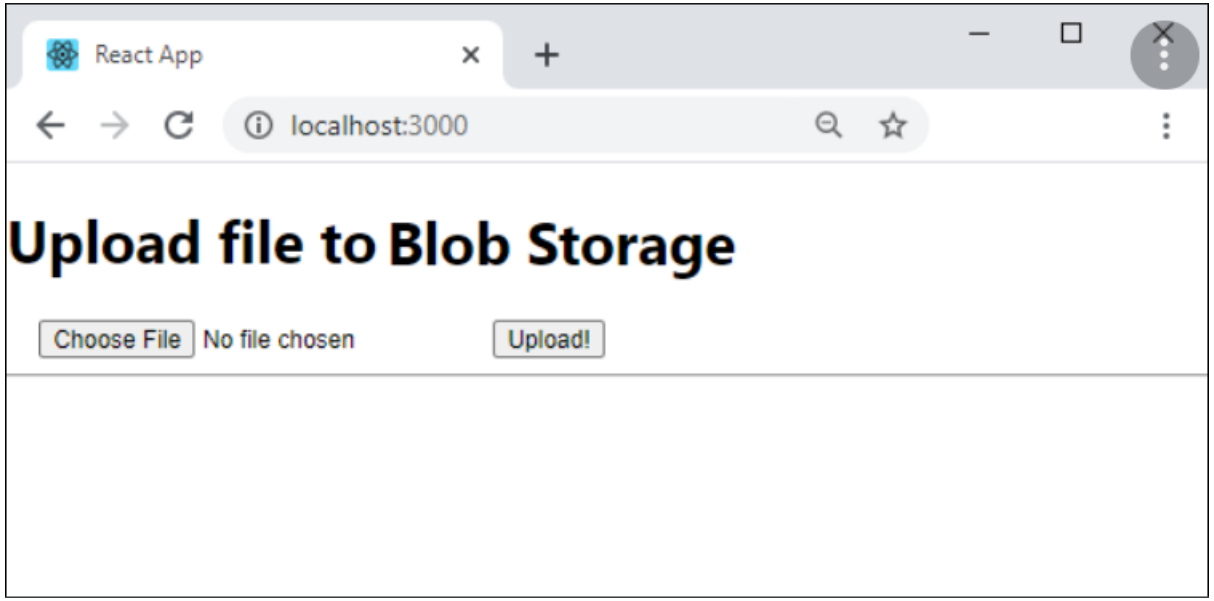
Patron de diseño Strategy

PROPOSITO:

Strategy es un patrón de diseño de comportamiento que “define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan”.

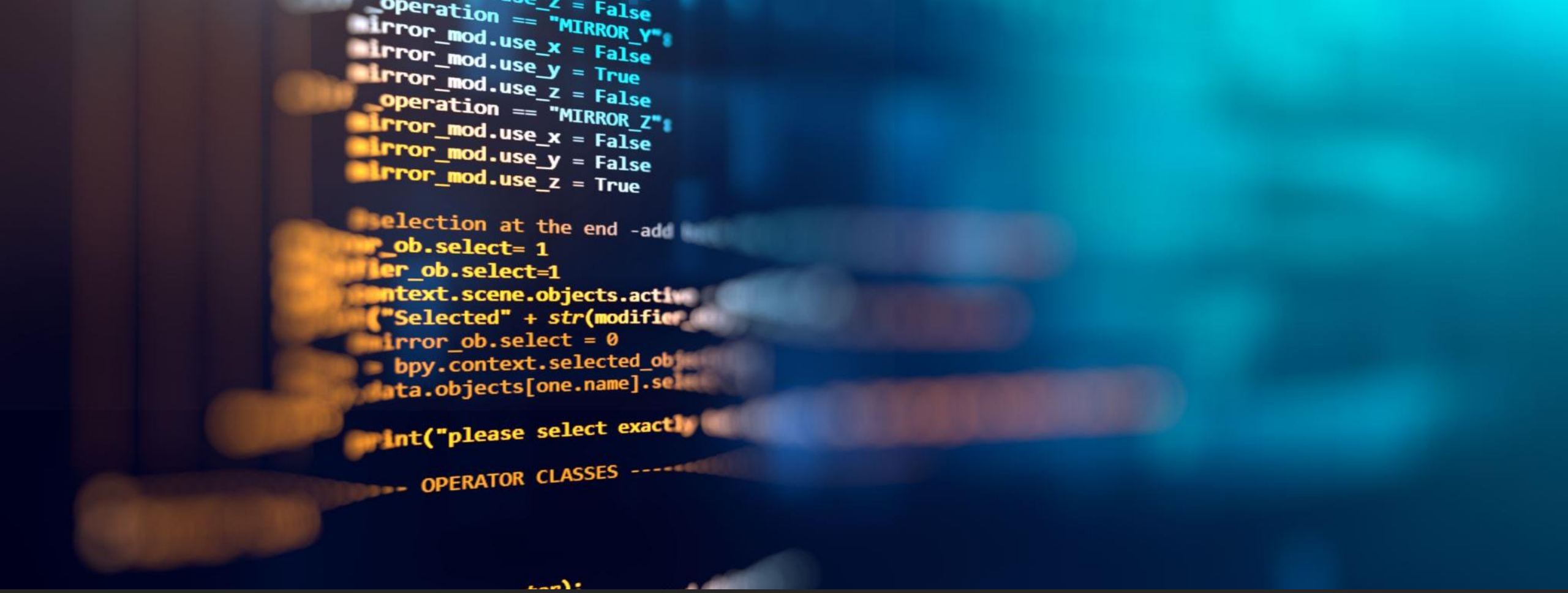
Patrones de diseño – Strategy

Problema: Aplicación para almacenar imágenes



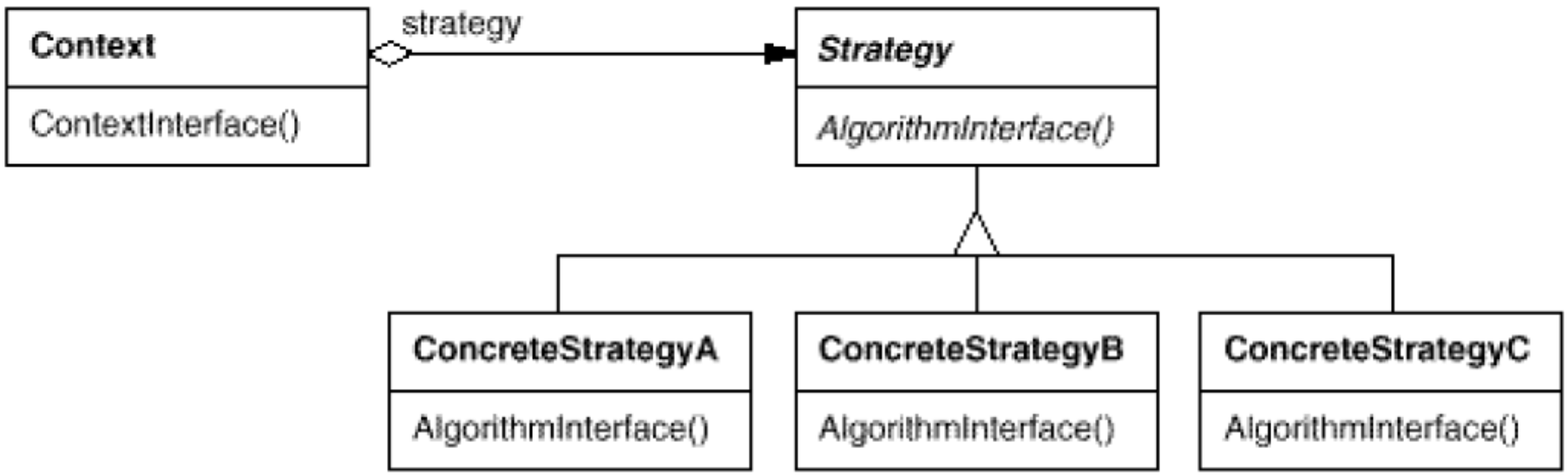
Algoritmos de Compresión:
JPEG, PNG, …

Filtros:
B&W, High Contrast



Demo

Revisemos en Código la implementación del problema

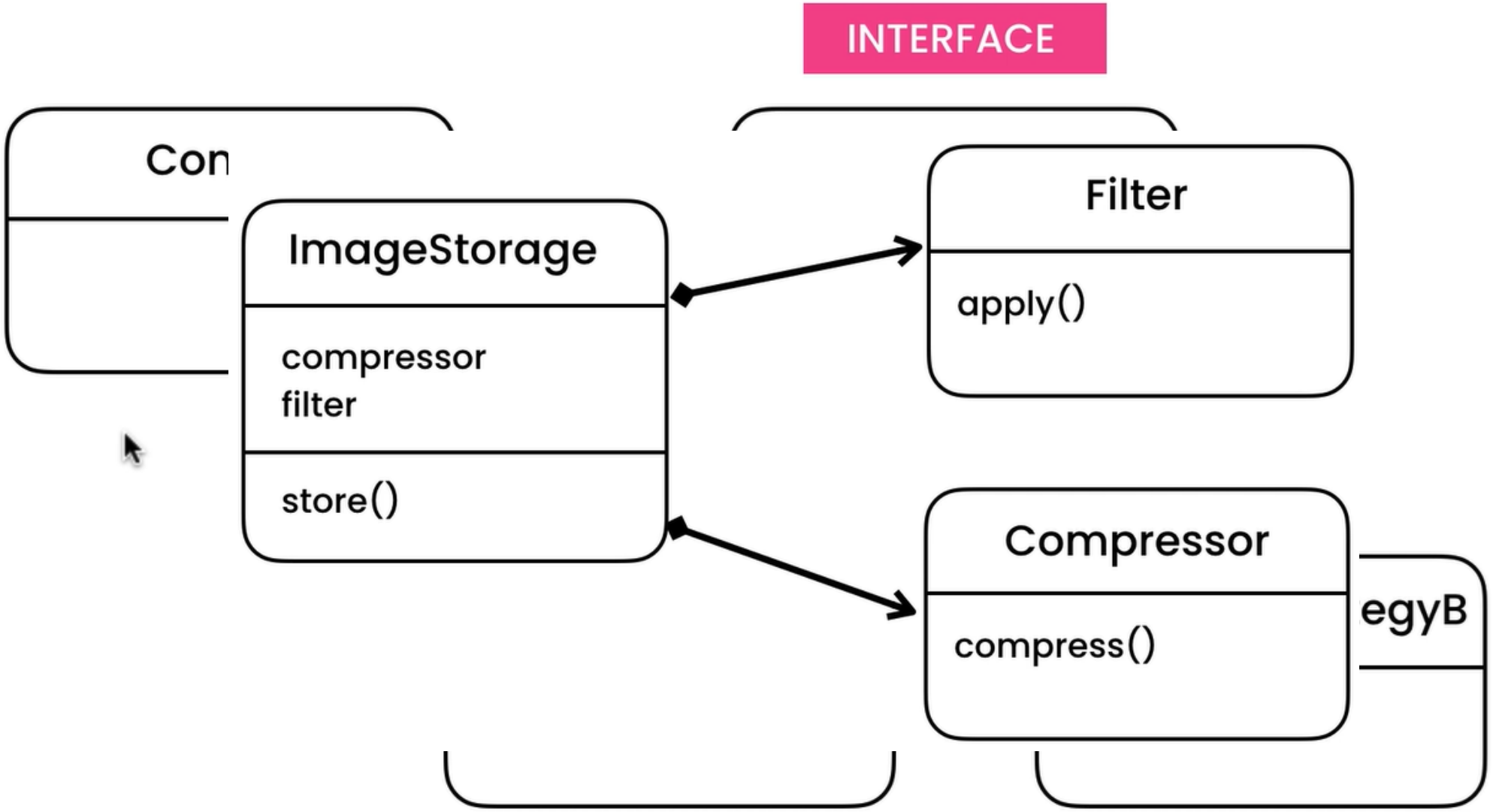


Patron de diseño - Strategy

Estructura

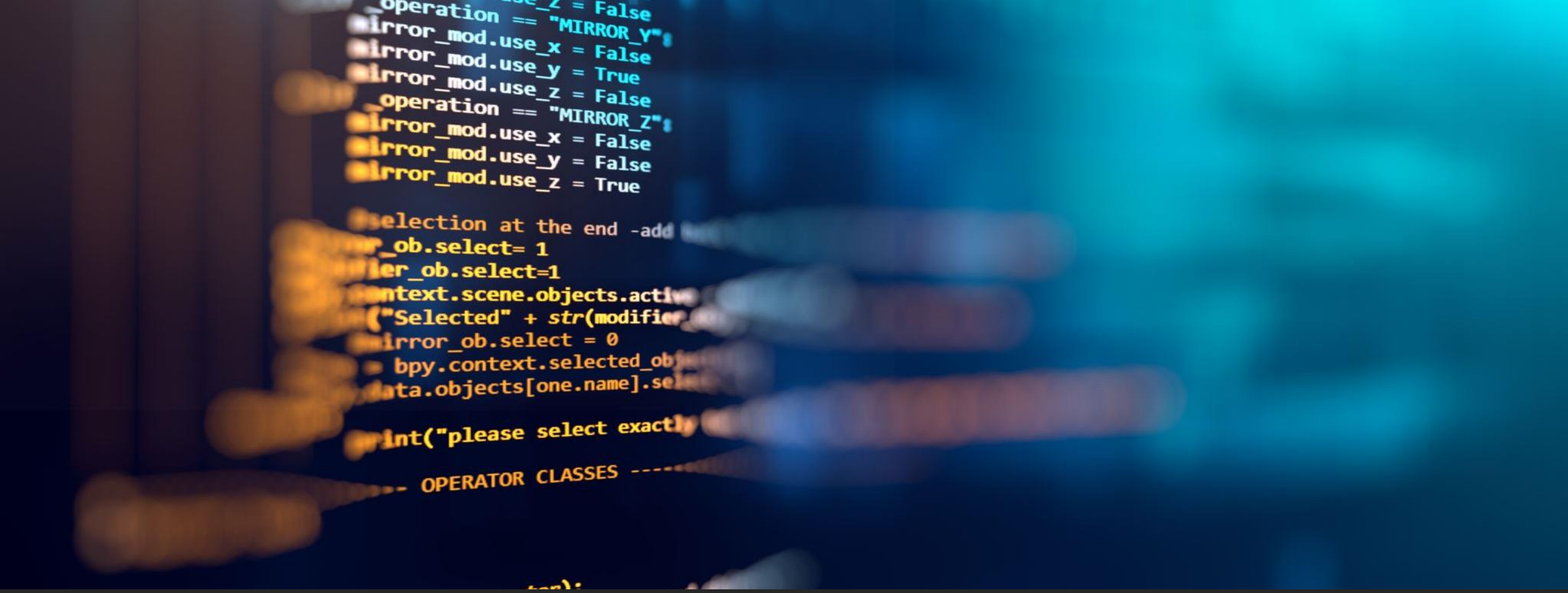
Patrones de diseño – Strategy

Implementación



Patrones de diseño – Strategy

Solución:



Demo

Revisemos en Código la implementación de la solución

Patrón de diseño Strategy

Aplicabilidad

Cuándo utilizar este patrón?

Utiliza el patrón Strategy cuando quieras utiliza distintas variantes de un algoritmo dentro de un objeto y poder cambiar de un algoritmo a otro durante el tiempo de ejecución.

Utiliza el patrón Strategy cuando tengas muchas clases similares que sólo se diferencien en la forma en que ejecutan cierto comportamiento.

Utiliza el patrón para aislar la lógica de negocio de una clase, de los detalles de implementación de algoritmos que pueden no ser tan importantes en el contexto de esa lógica.

Utiliza el patrón cuando tu clase tenga un enorme operador condicional que cambie entre distintas variantes del mismo algoritmo.

Patrones de diseño – Strategy

Ventajas:

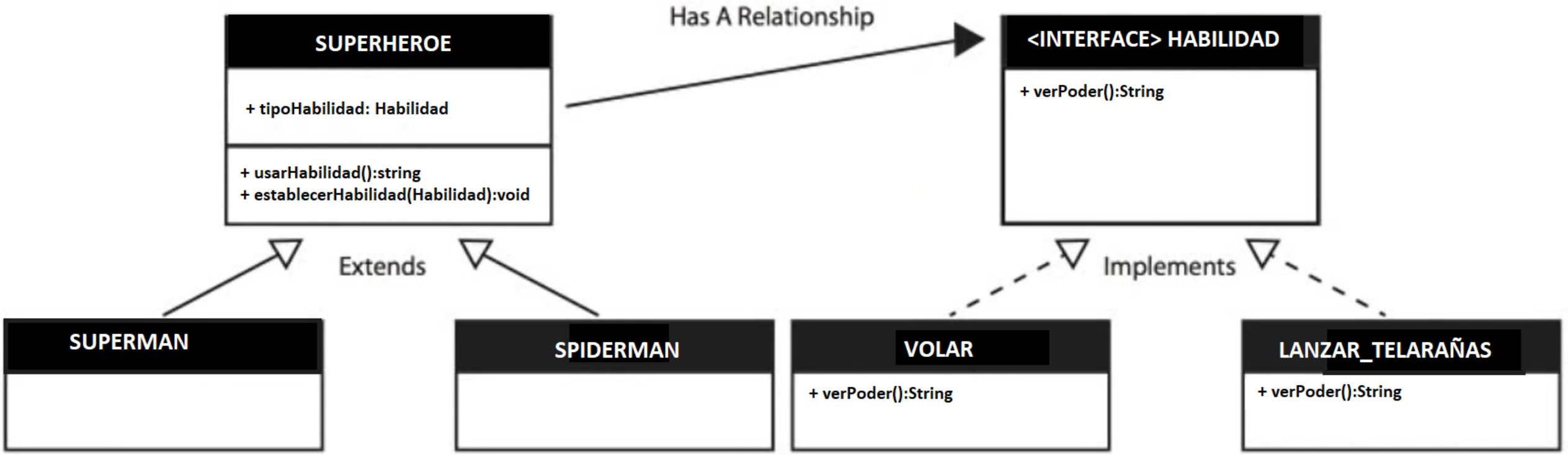
- ✓ Puedes intercambiar algoritmos usados dentro de un objeto durante el tiempo de ejecución.
- ✓ Puedes aislar los detalles de implementación de un algoritmo del código que lo utiliza.
- ✓ Puedes sustituir la herencia por composición.
- ✓ Principio de abierto/cerrado. Puedes introducir nuevas estrategias sin tener que cambiar el contexto.

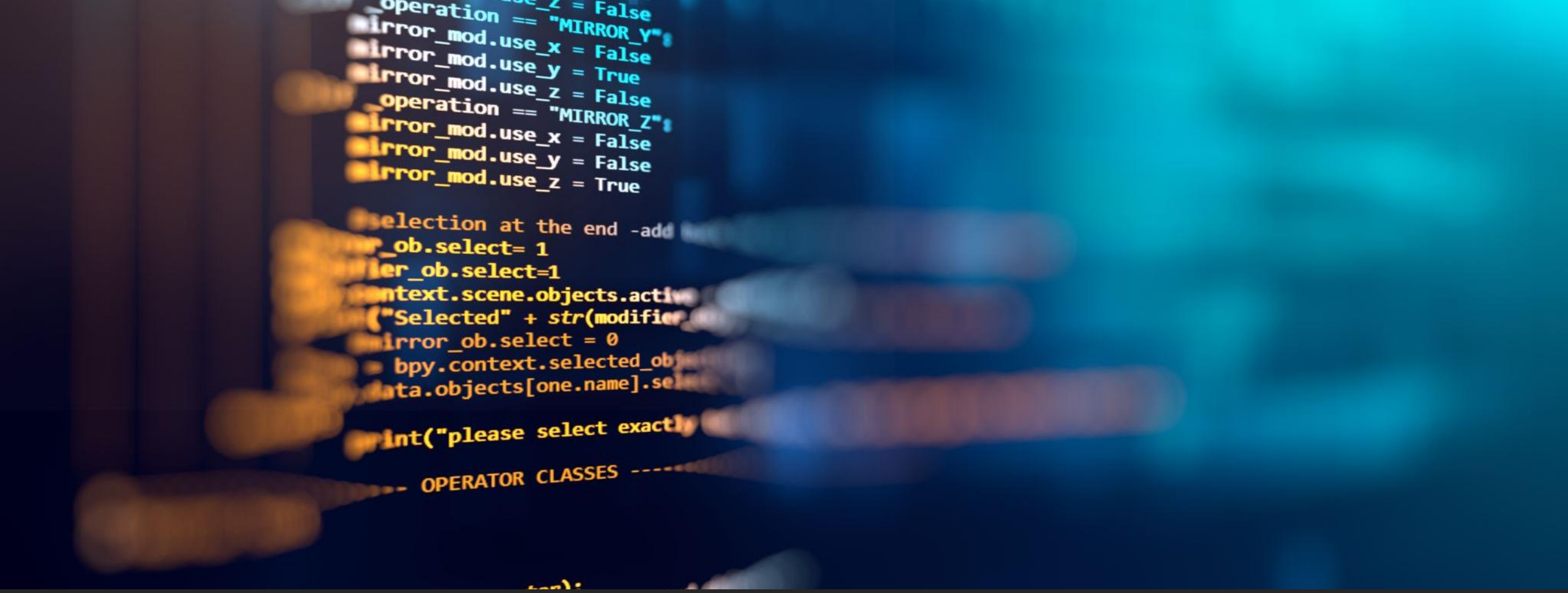
Desventajas:

- × Si sólo tienes un par de algoritmos que raramente cambian, no hay una razón real para complicar el programa en exceso con nuevas clases e interfaces que vengan con el patrón.
- × Los clientes deben conocer las diferencias entre estrategias para poder seleccionar la adecuada.
- × Muchos lenguajes de programación modernos tienen un soporte de tipo funcional que te permite implementar distintas versiones de un algoritmo dentro de un grupo de funciones anónimas. Entonces puedes utilizar estas funciones exactamente como habrías utilizado los objetos de estrategia, pero sin saturar tu código con clases e interfaces adicionales.

Patrones de diseño – Strategy

Práctica: Superheroes





Demo

Solución de la tarea

Patrones de diseño – Strategy

Tarea

Análisis de un Antivirus

- Método Analizar()

Antivirus Simple

- Iniciar()
- saltarZip
- detener()

Antivirus Avanzado

- Iniciar()
- analizarMemoria
- analizarKeyLoggers
- analizarRootKits()
- descomprimirZip()
- Detener()


Gracias