|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Diseño de sistemas | | |
| AREA | Tecnologías Aplicadas | REGIMEN | Semestral |
| MODALIDAD | Por examen final | CARGA HORARIA | |  | | --- | | 6 (seis) semanales,  90 totales | |

Se entrega la práctica en formato digital y papel. El formato digital debe tener el mismo encabezado que el presente texto. El formato en papel debe contener fecha, materia, curso, datos del alumno.

***Alumno:***

***Curso:***

***Fecha de entrega:***

**Tema: Patrones de diseño**

**Estrategia**

Encapsula algoritmos en clases, permitiendo que éstos sean re-utilizados e intercambiables. En base a un parámetro, que puede ser cualquier objeto, permite a una aplicación decidir en tiempo de ejecución el algoritmo que debe ejecutar.

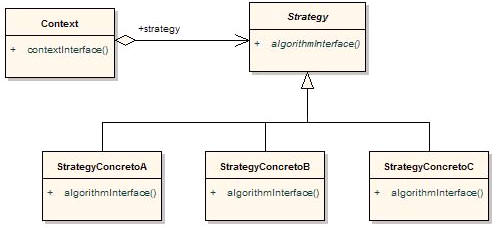
La esencia de este patrón es encapsular algoritmos relacionados que son subclases de una superclase común, lo que permite la selección de un algoritmo que varía según el objeto y también le permite la variación en el tiempo. Esto se define en tiempo de ejecución. Este patrón busca desacoplar bifurcaciones inmensas con algoritmos dificultosos según el camino elegido.

Este patrón debe utilizarse cuando:

* Un programa tiene que proporcionar múltiples variantes de un algoritmo o comportamiento.
* Es posible encapsular las variantes de comportamiento en clases separadas que proporcionan un modo consistente de acceder a los comportamientos.
* Permite cambiar o agregar algoritmos, independientemente de la clase que lo utiliza.

**Diagrama UML**

**Vista estática: diagrama de clases**



Strategy: declara una interfaz común a todos los algoritmos soportados.

StrategyConcreto: implementa un algoritmo utilizando la interfaz Strategy. Es la representación de un algoritmo.

Context: mantiene una referencia a Strategy y según las características del contexto, optará por una estrategia determinada.

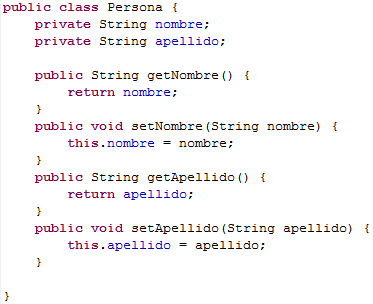
Context / Cliente: solicita un servicio a Strategy y este debe devolver el resultado de un StrategyConcreto.

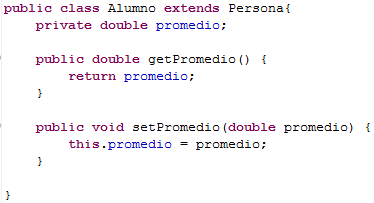
***Caso práctico a desarrollar en clase***

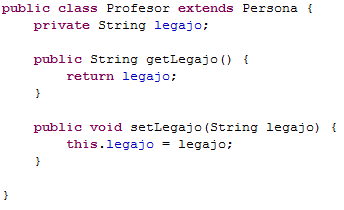
Imaginemos una biblioteca de un instituto educativo que presta libros a los alumnos y profesores. Imaginemos que los alumnos pueden asociarse a la biblioteca pagando una mensualidad. Con lo cual un libro puede ser prestado a Alumnos, Socios y Profesores.

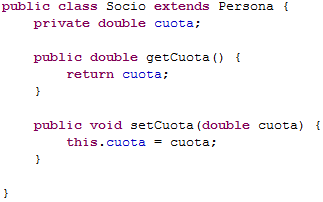
Por decisión de la administración, a los socios se les prestará el libro más nuevo, luego aquel que se encuentre en buen estado y, por último, aquel que estuviese en estado regular.

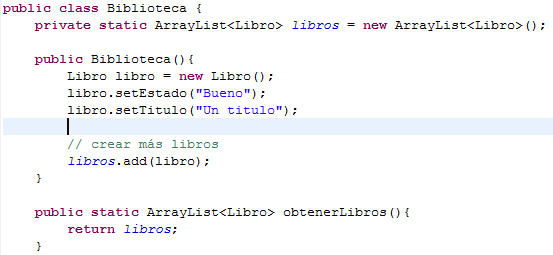
En cambio, si un Alumno pide un libro, ocurre todo lo contrario. Por último, a los profesores intentarán otorgargarles libros buenos, luego los recién comprados y, por último, los regulares. Este caso es ideal para el patrón Strategy, ya que dependiendo de un parámetro (el tipo de persona a la que se le presta el libro) puede realizar una búsqueda con distintos algoritmos.

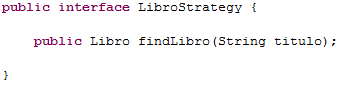


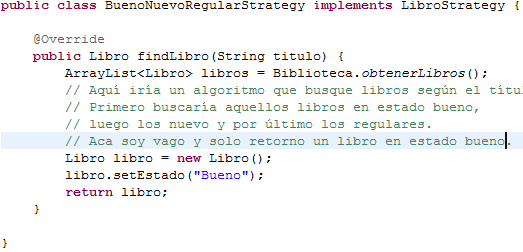


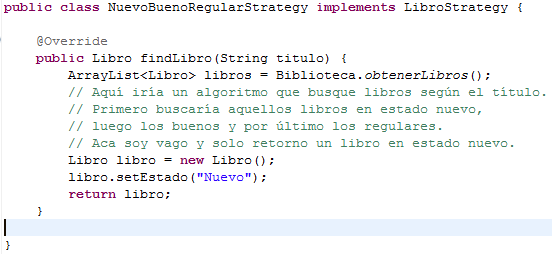


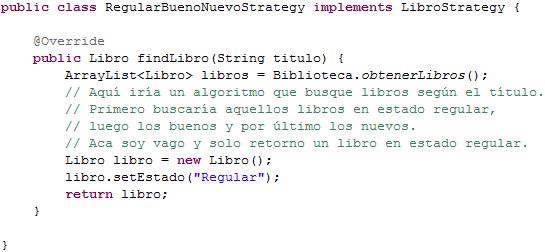


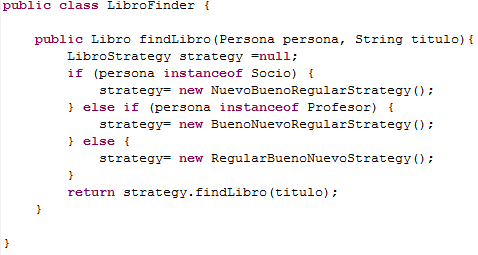


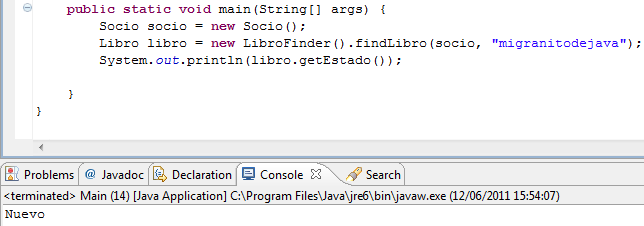












**Consecuencias**

* Permite que los comportamientos de los Clientes sean determinados dinámicamente sobre un objeto base.
* Simplifica los Clientes: les reduce responsabilidad para seleccionar comportamientos o implementaciones de comportamientos alternativos. Esto simplifica el código de los objetos Cliente eliminando las expresiones if y switch.
* En algunos casos, esto puede incrementar también la velocidad de los objetos Cliente porque ellos no necesitan perder tiempo seleccionado un comportamiento.