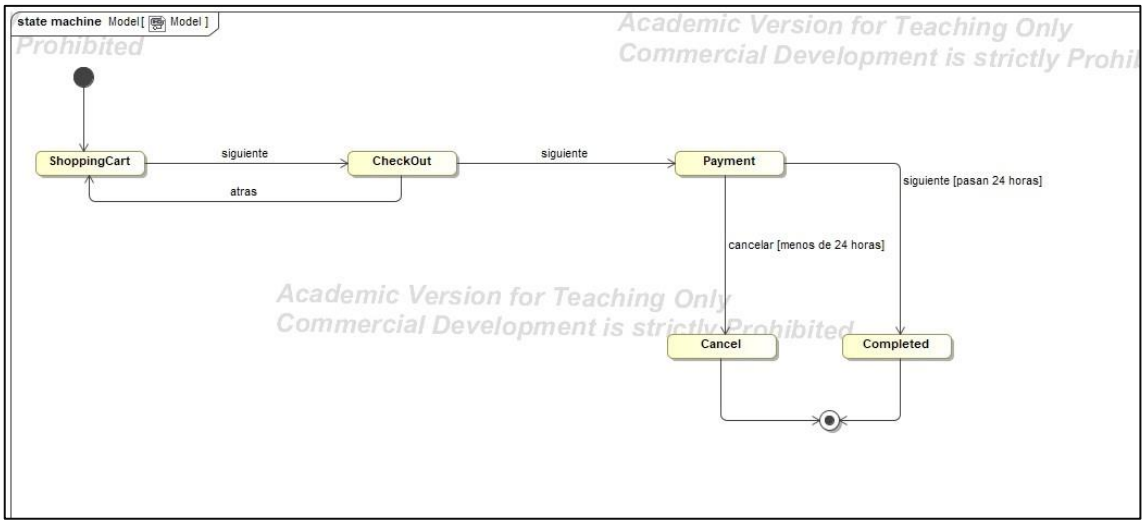
Sistema de compra online

En este ejercicio hemos decido usar el Patrón Estado, un patrón que viene perfecto ya que el comportamiento del sistema tiene que cambiar según el estado o momento de la compra. El patrón se basa de una interfaz base IOrderState, que se implementa a tantas clases estado como queramos (nosotros ShoppingCart,CheckOut,Payment…). Desde la clase principal Order se define un campo estado, y desde este hacemos las llamadas al estado actual, siempre usando la misma sintaxis ya que todas las funciones de todos los estados están definidas en IOrderState, pero no implementadas, dejando libertad para su implementación en cada estado especifico.

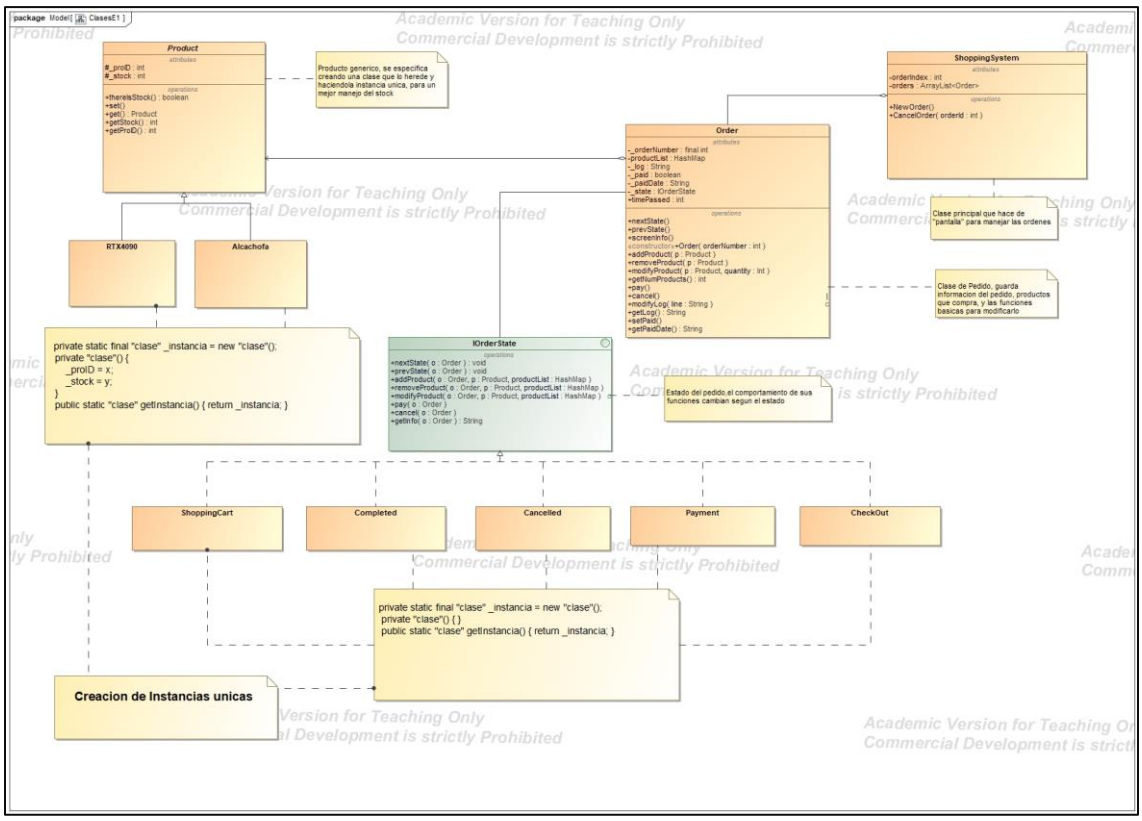
La clase central es la clase ShoppingSystem, que se encarga de guardar las Orders actuales y así como eliminarlas y crearlas, de manera controlada.

En cuanto a los Principios de Diseño empleados, el principio de Segregación de Interfaces no se cumple debido a la naturaleza del patrón, en cambio el principio de inversión de la dependencia sí, porque la interfaz IOrderState es de la que depende y se comunica con la clase Order, y no de las clases específicas de cada estado que podrían tener funciones diversas. Tampoco podemos afirmar que se cumpla el principio abierto-cerrado ya que el implementar un nuevo estado implica modificar la interfaz IOrderState, pero si podemos decir que se cumple el principio de responsabilidad única, ya que en cada estado el funcionamiento es obra de una única clase.

Hemos decidido que el mejor diagrama para representar el problema era sin duda un diagrama de Estados:



Y aquí el diagrama de clases:



Gestión de alertas de los parámetros de los tanques de un acuario

En este ejercicio hemos decidido emplear el patrón Observador, ya que es el que mejor adapta a las circunstancias debido a la naturaleza de eventos que suceden secuencialmente. El patrón consta en una clase abstracta Subject, que se hereda por clases que queremos observar, y una interfaz Observer, que es una extensión de aquellas clases que se observan, estas interactúan de manera que cuando el Subject cambie su estado (con un set por ejemplo), notificara a todos los observadores que estén suscritos a él, y estos ejecutaran su diversa función según su caso.

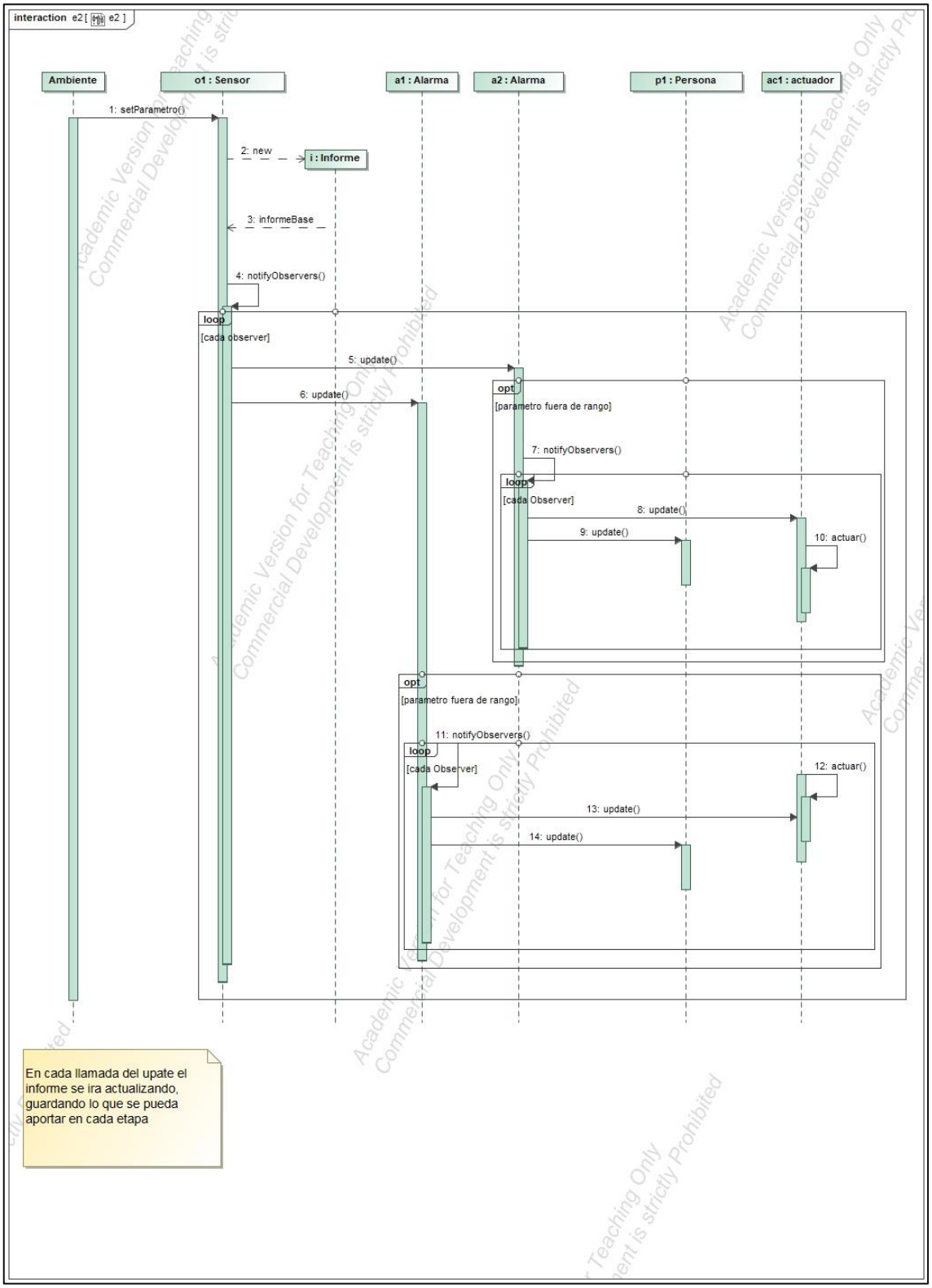
En nuestra implementación especifica, el sujeto le pasara por parámetro una clase adicional Informe, que guarda información del parámetro medido, nombre del tanque, tipo de alarma, etc., el informe se ira completando a medida que se hacen las llamadas, aportando cada clase la información que deba. Tenemos en total tres patrones observadores, donde la clase Alerta observa a la clase Sensor, que hace de sujeto, y la clase Alarma es a la vez Sujeto, observado

por la clase Persona y la clase Actuador, aun así, gracias al patrón, podemos implementar tantos sujetos y observadores como queramos, facilitando la modularidad, pero debemos tener en cuenta que el informe tendrá diferente información dependiendo por donde pase.

La clase central del programa es la clase Tanque, que se encarga de guardar los Sensores y Actuadores, y asignar a cada sensor un actuador.

En cuanto a los principios de diseño usados, se cumple el principio de responsabilidad única, ya que no existen clases Dios y cada clase tiene su función única por muy básica que sea (como la del sensor medir y ya), o el principio de abierto-cerrado, ya que se puede heredar la clase Alerta a cualquier otro tipo de alarma más especifico (como roja o naranja), y así especificar un comportamiento especifico (como que haga un ruido). Por último, decir que el Principio de sustitución de Liskov se cumple, ya que la clase Sujeto puede subscribir todo tipo de observadores sin importar la clase que sea (mientras implemente Obrserver), pero ya que las Alarmas solo pueden observar un Sensor, hemos tenido que a la hora de subscribir Alarmas se use un método attach modificado que use la clase Alarma en vez de la Observer, aun así, esto no afecta al principio.

Hemos decidido que le mejor diagrama dinámico para este problema seria un diagrama de secuencia:



Y aquí esta el diagrama de clases :