Informe Ejercicio 2

Patrones:

Para la implementación de este ejercicio solo se ha empleado un patrón de diseño, el patrón Observador. Este patrón se caracteriza por seleccionar una serie de objetos y crear la misma dependencia para cada uno de ellos, esto significa que cuando el objeto cambie de estado, todos sus objetos dependientes sean notificados y actualizados automáticamente.

Los objetos dependientes comparten una interfaz Observer entre ellos, esto hace que los Observers y el observado varíen de forma independiente. También permite añadir nuevas clases que implementen la interfaz Observer sin modificar el sujeto observado.

Hemos implementado en específico el patrón Observador con su modelo push ya que con nuestro update de la interfaz observers pasamos datos muy específicos necesarios para los observadores. El código creado permite crear cualquier tipo de parámetro a medir por el tanque. Además se podrían añadir más observadores sin problema aunque estemos usando el modelo push, pero esto significa que a lo mejor hay observadores que no usan todos los datos pasados por el update.

Para este ejercicio tenemos que llamar a dos Observers, Actuadores y Personal, y que cada uno de ellos haga ciertas cosas dependiendo de nuestra clase Alert que extiende una clase abstracta Subject.

La clase Alert crea una alerta con un nombre, un estado, un sensor con el que estar relacionado mediante una lista y cuatro valores float que crean dos intervalos, uno rojo y otro naranja que estará dentro del rojo. Esto permite comprobar a partir del parámetro del tanque en el update(float) de Alert ante qué tipo de alerta estamos, guardando ese tipo en nuestro estado.
Dependiendo del estado que tiene nuestra alerta es cuando el patrón Observador nos es útil ya que manda el tipo de alerta a todas las clases que tengan implementada la interfaz de Observer. En el caso de este ejercicio los Actuadores actualizarán el parámetro del tanque para que no cause una alerta roja o naranja y el Personal guardará un informe con todo el listado de alertas dadas y su tipo.

Nuestra alerta de tipo Alert tiene que estar guardada en una lista de alertas, asociada a un Sensor, por ello se crea una clase Sensor para que cada Sensor tenga varias alertas, pero que cada alerta esté relacionada con un único Sensor. A su vez este Sensor tiene que estar almacenado en una lista de sensores, asociada a un Tank, por ello se crea una clase Tank para que cada Tank tenga varios sensores, pero que cada sensor esté relacionada con un único Tank.

Principios de Diseño:

En los dos diagramas dados se puede ver que el código tiene una alta cohesión entre sus clases.
El Principio de inversión de la dependencia es la estrategia de depender de interfaces o clases y funciones abstractas, en vez de depender de clases y funciones concretas. En este código esto sucede con la clase abstracta Subject y la interfaz Observer. Aunque es cierto que existe cierta dependencia entre las clases Tank, Sensor y Alert ya que un objeto Sensor necesita estar relacionado con un clase Tank que tenga una lista de sensores Sensor y lo mismo pasa con Alert y Sensor.
El Principio de substitución de Liskov se puede ver entre las clases Subject y Alert, al no usarse override en Alert la clase abstracta Subject puede ser substituido por Alert.