**Justificaciones**

**Creando nuevos tipos de propiedades**

Decidimos implementar Factory Method, ya que permite definir una interfaz para crear un objeto, dejando la decisión de qué clase debe ser instanciada en manos de la subclase. Este patrón es el que mejor se adapta a nuestro diagrama de clases, permitiendo añadir propiedades nuevas solamente creando la clase de dicha propiedad heredando de la clase “propiedad”, además de declarar la relación de dependencia entre la nueva propiedad y la clase “fabricaConcreta”.

**Implementando auditoría y avisos en la aplicación**

Implementamos el patrón Observer, con el objeto “tablaEventos" que hereda de observador, el cual es subscriptor de “notificar”, de este modo “tablaEventos” recibe notificaciones de todos los eventos que realizan los observados. “tablaEventos” se encarga de guardar todos los eventos del sistema, que es capaz de ser consultado por el administrador además de esta también notificará al administrador vía correo electrónico. Este método permite que cuando un objeto cambie su estado, notificará al resto de objetos dependientes y estos se actualizarán automáticamente, la información llega a través de la instancia de un objeto llamado “Evento” el cual contiene los datos del mismo.

**Gestionando performance y unificando lógica**

Implementamos el patrón Singleton, ya que este permite la existencia de una única instancia del objeto en la aplicación, y debido a que se requiere unificar la lógica de avisos, este Patrón es la mejor opción. En nuestro diagrama esta implementado a través de la clase "singletonAvisos", y todo objeto que necesite hacer uso de la lógica de avisos lo hará a treves de una asociación directa con dicha clase.