**JUSTIFICACION USO DE PATRONES**

1. **Patrones Básicos.**
   1. Interface: No usado. Si bien el uso de interfaces me ayudaría a clasificar las clases según el uso de los métodos, es más eficiente para nuestro caso el uso de otros métodos que tienen acciones parecidas pero más definidas.
   2. Abstract Parent Class: Usado. Para nuestro ejercicio particular es muy funcional porque tengo objetos que heredan formas de hacer las cosas y características que son procesadas de forma diferente, además de que se integra con el uso de patrones creacionales que son más completos, para nuestro problema tengo preguntas, pero tienen formas diferentes de configurar sus posibles respuestas.
   3. Privates Methods. No usado. No usamos para este problema métodos privados ya que no hay procesos propios de alguna clase que sea solo accesible desde sí mismo.
   4. Accessor Methods: Usado. Es de utilidad ya que tengo atributos privados y por este motivo los métodos de acceso para editar o devolver la información de estos atributos son importantes para la comunicación entre clases.
   5. Constant Data Manager: No Usado. Para este programa no tengo datos de valor constante, por lo tanto el uso de un manager de datos constantes es innecesario.
   6. Immutable Object: No usado. No voy a tener objetos cuyo instanciación no pueda ser cambiada, por lo tanto no será usado el patrón de objetos inmutables.
   7. Monitor: No usado. No tendré objetos de uso sincronizado o por hilos.
   8. MVC: Usado. Es muy útil para nuestro caso, ya que puedo separar la implementación de la forma en que el usuario ve la información con un objeto controlando y comunicando las peticiones entre uno y el otro.
2. **Patrones Creacionales.**
   1. Abstract Factory: Usado. Tengo varios objetos parecidos, pero que haciendo ciertas acciones de forma diferente, preguntas de varias formas de definir la(s) respuesta(s) correcta(s), pero quiero que cada tipo de pregunta sea elaborada por un objeto aparte para que las respuestas definidas sean hechas para el tipo de pregunta en cuestión.
   2. Builder: Usado. Para elaborar un examen primero debo tener las preguntas hechas con su(s) respuesta(s) correcta(s) definidas, es así como builder me ayuda a realizar las cosas en el correcto orden para que la definición de un objeto examen sea correcto.
   3. Factory Method. No usado. Si bien puede ser una opción parecida al Abstract Factory, este no me permite hacer familias de preguntas con familia de respuestas y garantizar la entrega del tipo de respuesta correcta para el tipo de pregunta definida.
   4. Singleton: Usado. Para hacer conexión a la base de datos solo necesito un objeto instanciado de tipo conexion por ejecución.
   5. Prototype: No usado. No necesito de modelo de objetos instanciados para este problema en cuestión, el ideal es generar preguntas de forma aleatoria y en el caso de las preguntas con opción múltiple siempre será necesario cambiar, por lo menos en un orden diferente, por lo tanto prototype no es funcional.
3. **Patrones Estructurales.**
   1. Adapter: No usado. Como es de conocimiento este patrón realiza la comunicación entre dos interfaces convirtiendo la interfaz de una clase en una nueva interfaz (nueva versión) es decir adaptar una aplicación que ya se encuentra en uso y mejorarla a una nueva versión según lo que el cliente este solicitando. En el caso de nuestra aplicación no es necesario implementarla ya que estamos utilizando una única interfaz la cual tiene todos los componentes que el cliente necesita.
   2. Composite: pendiente.
   3. Facade: Usado. Para nuestra aplicación este patrón es funcional ya que se tomara el controlador como fachada para simplificar poder simplificar la comunicación entre el cliente y servidor. Es un beneficio ya que el cliente no tendrá que observar e interactuar con lo que está dentro de la aplicación, simplemente podrá observar una *interface* sencilla la cual podrá entender cómo manejarla.
   4. Flyweight: No usado. Actualmente nuestra aplicación no usara gran cantidad de información y el rendimiento de la memoria no se verá afectada por lo que no es necesario utilizar este patrón, ya que como se sabe lo que busca es reducir la redundancia de información idéntica liberando memoria para mejorar el rendimiento.
   5. Decorator: Usado. Es funcional para nuestra aplicación para agregar o elimnar de forma dinámica las funcionalidades de un objeto, esto lo implementamos para las preguntas ya que el usuario podrá ingresar nuevas opciones o a su vez eliminarlas, por lo que al realizar este proceso el contenido de las preguntas puede cambiar.