Explicaciones

1. **¿Cómo el Principio de Responsabilidad Única (SRP) hizo más fácil crear y mantener los validadores?**

Porque cada validador se enfoca solo en una cosa. Por ejemplo, uno valida que no haya campos vacíos, otro que el tipo de evento sea válido, otro que el producto tenga datos.
Si algo falla es más fácil ir a buscar dónde se encuentra el error para solucionarlo rápidamente.

2. ¿Cómo se aplica el Principio Abierto/Cerrado (OCP) en su diseño de clases de eventos y validadores?

El código está hecho para ser escalado sin modificar lo que ya está hecho.Por ejemplo, si quiero agregar un nuevo tipo de evento, solo creo una nueva subclase.Si quiero agregar una nueva validación, solo creo otro validador y lo pongo en la cadena.

3. ¿Qué ventajas ofrece el Patrón Factory Method en este escenario?

La ventaja es que el código sabe cómo construir el tipo correcto de evento automáticamente. El programa recibe los datos y el EventFactory lo clasifica si es ProductViewEvent o AddToCartEvent, dejando esa lógica a esa parte del código y no preocuparnos en el resto del código por ello.

4. ¿Cómo el Patrón Chain of Responsibility simplifica la lógica de validación secuencial?

Cada validador hace su parte y luego pasa el evento al siguiente. Los validadores están codificados de tal manera que puedan añadirse o removerse según sea necesario.

5. Si tuviéramos que almacenar estos eventos en una base de datos relacional (SQL) y una NoSQL (documentos), ¿cómo modelarían los datos de forma diferente en cada una?

En **SQL**, tendría que definir una tabla con columnas fijas (user_id, event_type, timestamp, etc.). Tal vez tendría una tabla base y otras específicas para cada tipo de evento.

En **NoSQL**, por ejemplo MongoDB, puedo guardar cada evento como un documento JSON, incluso con diferentes campos, lo que se adaptaría mejor al tipo de entrada de datos de este programa.