Actividad 2: Identificación y Justificación de Patrones de Diseño

1. **Patrón Creacional: Factory Method**
   Este patrón permite crear objetos sin tener que especificar la clase exacta del objeto. Este patrón se basa en una interfaz común y permite a las subclases definir que tipo de instancia devolver.
   - Aplicación:
     Dado que la plataforma maneja diferentes tipos de proyectos (ágiles, en cascada, híbridos) y módulos personalizados, se requiere una forma flexible de instanciar objetos sin acoplar el código a implementaciones concretas. Esto permite escalabilidad y facilita la incorporación de nuevos tipos de proyectos o módulos en el futuro sin modificar el código existente.

2. **Patrón Estructural: Facade**
   Este patrón proporciona una interfaz unificada y simplificada para un conjunto de interfaces en un subsistema. Además oculta la complejidad y reduce las dependencias directas entre los módulos.
   - Aplicación:
     La plataforma integrará múltiples servicios (gestión de tareas, predicción de retrasos con IA, generación de informes, integración con herramientas externas como Jira o Slack). Facade permitirá encapsular la lógica de integración y ofrecer una API clara y simplificada para los módulos de la aplicación, mejorando la mantenibilidad y reduciendo el acoplamiento entre componentes.

3. **Patrón de Comportamiento: Observer**
   El patrón Observer define una dependencia uno a muchos entre objetos, de manera que cuando un objeto cambia su estado, todos sus dependientes son notificados de forma automática.
   - Aplicación:
     En una plataforma de gestión de proyectos, múltiples usuarios se necesitará recibir actualizaciones en tiempo real sobre cambios en tareas, asignaciones, plazos o riesgos detectados por la IA. El patrón Observer permitirá que los módulos de notificaciones y dashboards escuchen cambios sin necesidad de un acoplamiento directo con el núcleo de la aplicación, facilitando la escalabilidad y mejorando la experiencia del usuario.