He elegido el **Patrón Estado (State Pattern)** para la gestión de los barcos en diferentes estados, y aquí te explico las razones clave:

- **Comportamiento Dependiente del Estado**

El patrón Estado es ideal para situaciones donde un objeto cambia su comportamiento dependiendo de su estado. En este caso, el comportamiento de un barco varía según si está en `DecommissionedState`, `SunkState`, o en algún otro estado. Cada uno de estos estados tiene sus propias reglas. Este patrón permite que cada estado se encargue de sus propias acciones sin complicar el resto del código con condicionales.

- **Evita Condicionales Complejos**

El patrón Estado encapsula el comportamiento de cada estado dentro de sus respectivas clases, evitando que el código principal se llene de condicionales complejos. Esto hace que el sistema sea mucho más limpio, modular y fácil de extender.

- **Facilidad para Añadir Nuevos Estados**

Con el patrón Estado, la incorporación de nuevos estados se hace de forma sencilla. Solo es necesario agregar una nueva clase que implemente la interfaz `ShipState` y definir su comportamiento específico. Este enfoque permite evolucionar el sistema de manera más flexible, sin necesidad de modificar clases existentes.

- **Desacoplamiento**

El patrón Estado desacopla el objeto principal (el barco) de la lógica específica de cada estado. Cada estado es responsable de sus propios comportamientos, lo que facilita la comprensión del código y su mantenimiento. Además, al no tener que gestionar manualmente las transiciones y el comportamiento de los estados dentro de la clase principal, se reduce el riesgo de errores.

- **Escalabilidad y Mantenibilidad**

El patrón Estado hace que el sistema sea escalable y fácil de mantener. Cuando se añaden nuevos estados o se modifican los existentes, no es necesario alterar las clases que gestionan el barco ni el flujo principal de operaciones. Solo es necesario crear o modificar las clases de los estados correspondientes.

Principios SOLID :

- **SRP (Responsabilidad Única)**: Cada clase tiene una única responsabilidad. El `Ship` se ocupa de representar al barco y sus estados, mientras que cada clase de estado maneja su propio comportamiento.
- **OCP (Abierto/Cerrado)**: El sistema está diseñado para que los nuevos estados puedan añadirse sin modificar las clases existentes. Solo se crean nuevas clases de estado que implementan la interfaz `ShipState`.
- **LSP (Sustitución de Liskov)**: Las clases de estado pueden ser sustituidas por cualquier clase que implemente la interfaz `ShipState`. Esto asegura que se puede cambiar un estado sin alterar el comportamiento del sistema.
- **ISP (Segregación de Interfaces)**: Cada clase de estado tiene una interfaz sencilla y específica que define únicamente las acciones que pueden realizarse en ese estado, lo que evita una interfaz demasiado cargada o innecesaria.
- **DIP (Inversión de Dependencias)**: La clase `Ship` depende de la interfaz `ShipState`, no de implementaciones concretas de los estados, lo que permite una mayor flexibilidad y facilita la extensión del sistema.