3 Hexagonal domain

El dominio es independiente de los detalles.

1. CONNECT

- ¿De quién depende la lógica de negocio?
- ¿Qué consecuencias tiene esta dependencia?

> Objetivo: entender la necesidad de proteger la lógica de negocio.

2. CONCEPT

Principios clave:

- **Domain**: núcleo con la lógica de negocio.
- **Details**: el resto de la solución.
- **Ports**: interfaces de comunicación.
- **Adapters**: implementaciones concretas.

Ideas fundamentales:

- La lógica de negocio es independiente de los detalles.
- El dominio expone sus necesidades de uso mediante ports.
- La solución implementa los ports mediante adapters.

3. CONCRETE PRACTICE

Partimos de capas de responsabilidad única y persistencia abstracta.

Aplicar el patrón **ports y adapters** en persistencia.

> Objetivo: patrón ports y adapters y familiaridad con hexagonal architecture.

- Capa de negocio
  - [ ] La capa `business` se renombra como `domain`
  - [ ] Las *interfaces* de `infrastructure` pasan a ser *ports* del `domain`.
  - [ ] Los *services* dependen de los *ports* en el constructor.

- Capa de persistencia
  - [ ] La capa `persistence` se renombra como `infrastructure`
  - [ ] Pierde sus *interfaces* y depende de los *ports* de `domain`.
  - [ ] Las implementaciones se quedan tal cual como *adapters* de los *ports*.
  - [ ] Se crea una *factoría* que proporciona *adapters*.

- Capa de presentación
  - [ ] No es necesario renombrar la capa `presentation` por el momento
  - [ ] Depende de la capa de `domain` y de la *factoría* de `infrastructure`.
  - [ ] Usa la *factoría* de `infrastructure` para obtener *adapters*.
  - [ ] Envía los *adapters* a los *services* de `domain`.

4. CONCLUSIONS

- Los patrones aportan claridad y consistencia.
- El objetivo es fomentar la flexibilidad y *testability*.
- La inversión del control permite la independencia de la lógica de negocio.
- ¿Cómo de independiente es tu lógica de negocio?