Plan de trabajo por equipos con dependencias
==========
(Desde “Organización en 3 equipos” hasta “Anti-patrones a evitar”, actualizado con
qué puede hacer cada equipo en cada fase SIN esperar y qué requiere esperar.)

Glosario súper breve (para todos)

- **Contrato (interface)**: lista de métodos que otros equipos pueden usar. Define la “forma” de hablar entre capas.
- **Repositorio (Repository)**: interface que define cómo guardamos y leemos datos (guardar, buscar, listar).
- **Servicio (Service)**: orquesta reglas de negocio usando repositorios (ej.: crear/cancelar/reprogramar turnos).
- **CLI**: interfaz de consola; menú que interactúa con el usuario.
- **Seed**: datos de ejemplo para probar rápido (pacientes, profesionales, etc.).
- **ADR**: nota corta explicando por qué se eligió una decisión técnica.

Semáforo de dependencias (leyenda)

- 🟢 Puede avanzar **ya**, sin esperar a nadie.
- 🟡 Puede avanzar **con stubs/mocks** (estructuras mínimas que compilan). Ideal para no frenarse.
- 🔴 **Debe esperar** un hito específico. Indicamos quién y por qué.

Organización en 3 equipos
==========

Ramas sugeridas: main (estable) · dev (integración) · trabajo diario en team-1/*, team-2/*, team-3/* + PRs a dev.
Convención commits: feat|fix|test|docs|refactor|chore: descripción.

Equipo 1 — Dominio & Contratos

Objetivo: definir el modelo del negocio y los contratos (interfaces) que usarán los otros equipos. Sin dependencias externas.

Paquetes / Carpetas
```
src/main/java/com/…/turnos/
  ├─ domain/          # entidades e invariantes
  │   ├─ Clinica.java
  │   ├─ Sucursal.java
  │   ├─ Profesional.java
  │   ├─ Paciente.java
  │   ├─ Especialidad.java
```

```
    │   └─ Turno.java
    ├─ domain/value/      # value objects
    │   ├─ DNI.java
    │   ├─ Email.java
    │   └─ Id.java
    ├─ validation/        # validadores de dominio
    │   ├─ TurnoValidator.java
    │   └─ PacienteValidator.java
    ├─ exception/         # excepciones de dominio
    │   ├─ DomainException.java
    │   ├─ ValidationException.java
    │   └─ NotFoundException.java
    └─ repository/        # CONTRATOS (interfaces)
        ├─ TurnoRepository.java
        ├─ PacienteRepository.java
        └─ ProfesionalRepository.java
```

Contratos mínimos (métodos guía)

```
 interface TurnoRepository {
   Turno guardar(Turno turno);
   Optional<Turno> porId(Id id);
   List<Turno> porDiaYSucursal(LocalDate fecha, Id sucursalId);
   List<Turno> porProfesionalYDia(Id profesionalId, LocalDate fecha);
   boolean existeSolapamiento(Id profesionalId, LocalDateTime inicio,
Duration duracion);
   void eliminar(Id id);
 }
```

Entregables

- Entidades con invariantes (p. ej., un Turno no puede iniciar en el pasado; sin solapes).
- *Repository definidos y documentados (JavaDoc) con ejemplos de uso.
- Tests unitarios del dominio (sin persistencia).

Criterios de aceptación

- Los tests de dominio pasan (./gradlew test).
- Ninguna clase en domain depende de service, persistence o app.
- Interfaces publicadas permiten a Equipo 2 y 3 trabajar sin bloquearse.

Equipo 2 — Persistencia & Config (Adaptadores)

Objetivo: implementar los repositorios definidos por Equipo 1. Empezar en memoria, luego CSV/JSON, y después JDBC (H2/SQLite).

Paquetes / Carpetas

```
 src/main/java/com/.../turnos/
   └─ persistence/
```

```
        ├─ memory/
        │   ├─ InMemoryTurnoRepository.java
        │   ├─ InMemoryPacienteRepository.java
        │   └─ InMemoryProfesionalRepository.java
        ├─ file/
        │   ├─ CsvTurnoRepository.java
        │   └─ JsonTurnoRepository.java
        └─ jdbc/
            ├─ DataSourceFactory.java
            └─ JdbcTurnoRepository.java
  src/main/resources/
    ├─ data-ejemplo/*.csv|json
    └─ sql/ (cuando llegue JDBC)
  config/app.properties         # rutas de archivos, flags, etc.
```

Entregables por iteración
  1. Iteración 1 (ahora): repos in-memory 100% funcionales + datos semilla.
  2. Iteración 2: repos file (CSV/JSON) con lectura/escritura atómica y validaciones.
  3. Iteración 3: JDBC (H2/SQLite) con DataSourceFactory, DDL y pruebas de integración.

Criterios de aceptación
  - Implementaciones pasan los tests de Equipo 3 que usan los repos.
  - persistence/* no depende de app; solo de domain/repository.
  - Para file/JDBC: manejo de errores (IO/SQL) con excepciones claras y logs.

Equipo 3 — Servicios & Interfaz (CLI)

Objetivo: implementar casos de uso sobre los contratos de repos y entregar una CLI utilizable.

Paquetes / Carpetas

```
  src/main/java/com/…/turnos/
    ├─ service/
    │   ├─ TurnoService.java        # crear/listar/cancelar/reprogramar
    │   ├─ PacienteService.java
    │   └─ AgendaService.java
    ├─ app/
    │   ├─ Main.java               # entry point
    │   ├─ CliMenu.java             # menú principal
    │   └─ ConsoleIO.java           # util de entrada/salida
    ├─ dto/
    │   └─ TurnoDTO.java
    ├─ mapper/
    │   └─ TurnoMapper.java
```

```
    └─ util/
       └─ DateTimeUtils.java
```

Contratos de servicio (guía)

```
class TurnoService {
  Turno crearTurno(Id sucursalId, Id profesionalId, Id pacienteId,
                   LocalDateTime inicio, Duration duracion);
  List<Turno> listarPorDiaYSucursal(LocalDate fecha, Id sucursalId);
  void cancelarTurno(Id turnoId);
  Turno reprogramarTurno(Id turnoId, LocalDateTime nuevoInicio);
}
```

Entregables

- CLI con opciones: Crear / Listar / Cancelar / Reprogramar (por clínica/sucursal).
- Tests de servicio y e2e contra repos in-memory.
- application configurado (mainClass) y scripts de ejecución.

Criterios de aceptación

- ./gradlew run permite la operación básica por consola.
- Servicios usan solo interfaces de repos (no implementaciones concretas).
- Tests de servicio cubren reglas de negocio y errores esperables.

Dependencias entre equipos (acuerdo)

Equipo 3 (Servicios/CLI) --> usa --> repository.* (definido por Equipo 1)
Equipo 2 (Persistencia) --> implementa --> repository.* (Equipo 1)
Equipo 1 (Dominio) --> no depende de 2 ni 3 (núcleo independiente)

Backlog por iteración (sugerido)

Iteración 0 (hoy)
- Equipo 1: publicar entidades + TurnoRepository (firmas definitivas) + tests.
- Equipo 2: InMemory*Repository con datos semilla.
- Equipo 3: TurnoService mínimo + Main + CliMenu con Crear y Listar.

Iteración 1
- Equipo 1: reglas extra (cancelación, reprogramación, ventanas de atención).
- Equipo 2: Repos file (CSV/JSON) + config de rutas.
- Equipo 3: CLI completa (Cancelar/Reprogramar + filtros) + tests e2e.

Iteración 2
- Equipo 2: JDBC (H2/SQLite) + migraciones SQL + pruebas integración.
- Equipo 3: logging, manejo de errores amable en CLI, reporte

simple.
    - (Opcional) Preparar módulo REST o UI según avance.

Roles de revisión y ramas
  Code owners
    - Equipo 1 → domain/*, validation/*, exception/*, repository/*
    - Equipo 2 → persistence/*, config/*, resources/data-*,
resources/sql/*
    - Equipo 3 → service/*, app/*, dto/*, mapper/*, util/*
  Ramas de trabajo
    - Equipo 1: team-1/domain-contratos-*
    - Equipo 2: team-2/persistencia-*
    - Equipo 3: team-3/servicios-cli-*
  Definition of Done (DoD)
    - Build y tests pasan (./gradlew clean test).
    - Sin TODO críticos.
    - JavaDoc en interfaces y servicios públicos.
    - README/CHANGELOG actualizado si aplica.

Scripts de verificación (para todos)
  scripts/verificar-entorno.sh
    java -version
    ./gradlew --version
    ./gradlew clean test

Semáforo por fase y pasos detallados (con dependencias explícitas)
==========

FASE 0 — Kick-off y base de proyecto (Día 0–0.5)

Objetivo: tener el proyecto listo para compilar, testear y correr
“Hola”.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA.
  Qué hacer:
    - Escribir docs breves: arquitectura general y nombres de
paquetes.
  Por qué no espera:
    - No necesita de otros; son acuerdos de nombre y estructura.

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar YA.
  Qué hacer:
    - Configurar Gradle (Java 21 toolchain), JUnit 5, .gitignore y
script de verificación.

Por qué no espera:
- Build y testing no dependen de dominio todavía.

Equipo 3 — Servicios & CLI
Estado: Puede avanzar YA.
Qué hacer:
- README inicial, clase Main mínima, `application.mainClass`, imprimir “Hola”.
Por qué no espera:
- La CLI puede nacer vacía y crecer después.

FASE 1 — Publicación de contratos (Día 0.5–1)

Objetivo: publicar interfaces y entidades mínimas para destrabar a todos.

Equipo 1 — Dominio & Contratos
Estado: Puede avanzar YA.
Qué hacer:
- Crear POJOs básicos (Clinica, Sucursal, Profesional, Paciente, Especialidad, Turno).
- Crear value objects (Id, DNI, Email), validadores y excepciones base.
- Publicar interfaces repository/* con firmas definitivas y JavaDoc.
- Tests de dominio (sin persistencia).
Nota:
- Este “esqueleto estable” es la clave para que 2 y 3 avancen sin trabas.

Equipo 2 — Persistencia & Config
Estado: Puede avanzar con STUBS mientras Equipo 1 publica contratos.
Qué hacer:
- Crear clases InMemory*Repository que por ahora lancen UnsupportedOperationException o devuelvan vacío.
- Preparar recursos y config (app.properties) sin lógica aún.
¿Cuándo tendría que esperar ()?
- Solo si Equipo 1 demora en **publicar las interfaces** (métodos y tipos).

Equipo 3 — Servicios & CLI
Estado: Puede avanzar con STUBS mientras Equipo 1 publica contratos.
Qué hacer:
- Stubs de servicios que llamen a repos vacíos + menú de navegación.
¿Cuándo tendría que esperar ()?
- Solo si Equipo 1 demora en **publicar las interfaces**.

FASE 2 — MVP en memoria (Día 1–2)

Objetivo: flujo Crear/Listar funcionando con repos en memoria.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA.
  Qué hacer:
    - Afinar invariantes (sin solapes por profesional y horario;
duración válida).
    - Extender validadores y mensajes claros.

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar YA (usa interfaces publicadas).
  Qué hacer:
    - Implementar InMemory*Repository real con estructuras
Map/índices.
    - Script de seed con datos de ejemplo.
    - Tests unitarios de repos.
  No espera a:
    - Equipo 3: puede desarrollar en paralelo; compartirán contract
tests.

Equipo 3 — Servicios & CLI
  Estado: Puede avanzar YA (con repos in-memory).
  Qué hacer:
    - Implementar TurnoService (crear/listar) y conectar CLI.
    - Tests de servicio y E2E contra in-memory.
  No espera a:
    - Equipo 2: si aún no está listo, puede usar **mocks** temporales.

FASE 3 — Persistencia en archivo (CSV/JSON) (Día 2–3)

Objetivo: guardar/cargar desde disco sin DB.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA.
  Qué hacer:
    - Definir estados del turno: PENDIENTE, CONFIRMADO, CANCELADO.
    - ADR “Archivo primero” (por simplicidad).

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar YA.
  Qué hacer:
    - Implementar Csv* o Json* repository.
    - Lectura/escritura atómica (archivo temporal + rename).
    - Config conmutables: persistence=memory|file.
    - Tests de integración con archivos temporales.

No espera a:
- Equipo 3: usará la capa por **interfaces**.

Equipo 3 — Servicios & CLI
Estado: Puede avanzar YA.
Qué hacer:
- AppConfig para seleccionar persistence por config.
- Agregar Cancelar y Reprogramar a la CLI.
- Mensajes de error claros.
No espera a:
- Implementación real de archivo si aún no está: seguir en memory y luego conmutar.

FASE 4 — JDBC (H2/SQLite) (Día 3–5)

Objetivo: persistencia robusta con SQL.

Equipo 1 — Dominio & Contratos
Estado: Puede avanzar YA.
Qué hacer:
- Proponer modelo relacional (tablas, FKs, índices y constraint UNIQUE anti-solape).
- ADR “SQLite/H2” (embebidas, simples).

Equipo 2 — Persistencia & Config
Estado: Puede avanzar YA.
Qué hacer:
- DataSourceFactory, Jdbc*Repository, DDL y seeds.
- Transacciones, PreparedStatement, manejo de SQLException.
- Tests de integración con H2 en memoria.
No espera a:
- Equipo 3: seguirá usando interfaces; después cambia config a jdbc.

Equipo 3 — Servicios & CLI
Estado: Puede avanzar usando memory/file y luego conmutar a jdbc.
Qué hacer:
- Agregar filtros y reporte simple.
- Soportar import/export (opcional).
¿Cuándo espera ()?
- Solo si necesita probar **sí o sí** contra JDBC real antes de que 2 publique.

FASE 5 — Reglas de negocio avanzadas (Día 5–6)

Objetivo: políticas de reprogramación/cancelación y ventanas de atención.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA.
  Qué hacer:
    - Políticas (p.ej., reprogramar solo ≥24h antes; estados con auditoría).
    - Tests de dominio para casos límite.

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar ajustando consultas y constraints.
  Qué necesita:
    - Reglas finales de Equipo 1 (para reflejar en DB/validaciones).
  Nota:
    - Normalmente no bloquea; se puede avanzar con supuestos y luego ajustar.

Equipo 3 — Servicios & CLI
  Estado: Puede avanzar implementando mensajes y bloqueos UX.
  Qué necesita:
    - Conocer reglas de Equipo 1 para mostrar mensajes correctos.

FASE 6 — Logging, errores y DX (Día 6)

Objetivo: trazas claras, errores amables y modo verbose.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA (mejorar mensajes de excepciones).

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar YA (logs INFO/WARN/ERROR y sanitización de datos).

Equipo 3 — Servicios & CLI
  Estado: Puede avanzar YA (manejo de excepciones y flag --verbose).

FASE 7 — Calidad, estilo y cobertura (Día 6–7)

Objetivo: calidad consistente en todo el repo.

Todos los equipos
  Estado: Pueden avanzar YA.
  Qué hacer:
    - Checkstyle, SpotBugs, Jacoco (cobertura).
    - JavaDoc pública, limpieza de warnings y TODOs.
  No espera a:
    - Nadie: es transversal.

FASE 8 — Empaquetado y entrega (Día 7)

Objetivo: instalación y ejecución sin IDE.

Equipo 2 — Persistencia & Config
  Estado: Puede avanzar YA.
  Qué hacer:
    - ./gradlew installDist (binarios), config externa fuera del JAR.

Equipo 3 — Servicios & CLI
  Estado: Puede avanzar YA.
  Qué hacer:
    - Script run.sh, soporte --config=/ruta/app.properties, README
final.

Equipo 1 — Dominio & Contratos
  Estado: Puede avanzar YA.
  Qué hacer:
    - CHANGELOG 1.0.0 y docs de arquitectura actualizadas.

FASE 9 — (Opcional) REST o UI (Día 8+)

Variante REST (Spring Boot)
  - Nuevo módulo api-rest/ con controladores.
  - Reusar services y repositories existentes.
  - Tests @WebMvcTest / @SpringBootTest (perfil H2) y OpenAPI.

Variante UI (JavaFX)
  - Módulo ui-javafx/ con escenas (listado, creación, filtros).
  - Reusar services ya probados.

Anti-patrones a evitar
==========
- Acoplar service a una **implementación** de repo (siempre a la
interfaz).
- Meter lógica de negocio dentro de **CLI** o de **Repository** (debe
ir en **service/domain**).
- Saltarse tests o logs: genera bugs difíciles de rastrear.
- Mezclar formato de archivo con reglas de dominio: **separar capas**.