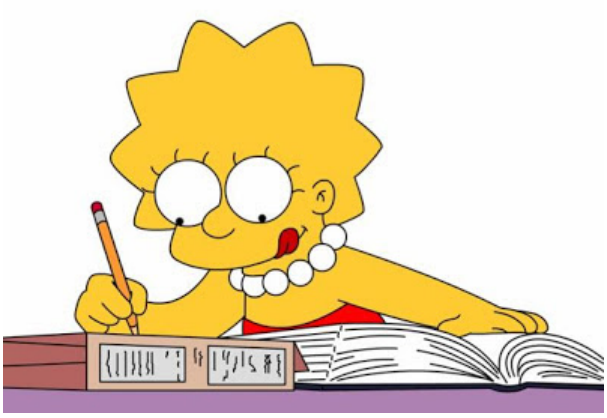
Ejercicios de repaso



Este proyecto reúne **ejercicios de repaso** de todo lo que hemos visto en clase durante este **trimestre**.

Estructura de paquetes

- `src/main/java/com/docencia/<tema>/ejercicioN` → código del ejercicio (`EjercicioN.java`) y, cuando aplique, clases del dominio.
- `src/test/java/com/docencia/<tema>/ejercicioN` → tests `EjercicioNTest.java` para verificar el correcto funcionamiento.

Temas incluidos

En los paquetes anteriores se sigue la estructura de los distintos temas que hemos venido trabajando en clase estos meses:

- Fundamentos: `com.docencia.condiciones`, `com.docencia.arrays`, `com.docencia.clases`, `com.docencia.mixto`
- Simulacros: `com.docencia.listas`, `com.docencia.composicion`, `com.docencia.herencia`

Ejecutar los tests

Si quieres ver cómo funcionan los tests, ejecuta en el directorio del proyecto:

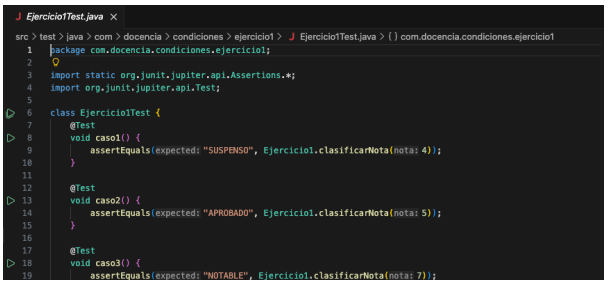
```
jpexposito@MacBook-Air-de-Joatham ejercicios-repaso % ls -la
total 120
drwxr-xr-x@  9 jpexposito  staff    288 13 dic 11:35 .
drwx------@ 52 jpexposito  staff   1664 13 dic 11:40 ..
-rw-r--r--@  1 jpexposito  staff   6148 13 dic 11:07 .DS_Store
-rw-r--r--@  1 jpexposito  staff     32 13 dic 11:05 .gitignore
drwxr-xr-x@  3 jpexposito  staff     96 13 dic 11:35 images
-rw-r--r--@  1 jpexposito  staff   1278 13 dic 11:05 pom.xml
-rw-r--r--@  1 jpexposito  staff  42122 13 dic 11:46 README.md
drwxr-xr-x@  5 jpexposito  staff    160 13 dic 11:07 src
drwxr-xr-x@  8 jpexposito  staff    256 13 dic 11:32 target
```

Ejecuta:

```
mvn clean test
```

> Aviso: muchos ejercicios están planteados como **esqueletos** con `TODO` y/o `UnsupportedOperationException`.
> La idea es que realices los ejercicios hasta que **los tests pasen**.

Puedes ir ejecutando los tests de cada ejercicio:



> **Importante**: Si te queda ***alguna*** duda en la **descripción del ejercicio, *visualiza el test correspondiente*** para **interpretar los parámetros de entrada**, y la **respuesta esperada**.

Orden recomendado de desarrollo (de menor a mayor dificultad)

1. **Condiciones y `switch`** → `com.docencia.condiciones.ejercicio1..10`
2. **Bucle `for` sobre arrays** → `com.docencia.arrays.ejercicio1..10`
3. **Construcción de clases** (`constructor vacío`, `constructor con id`, `equals/hashCode/toString`) → `com.docencia.clases.ejercicio1..10`
4. **Mixtos (arrays + condiciones/switch + clase)** → `com.docencia.mixto.ejercicio1..20`
5. **Listas + Composición + Herencia** (simulacros) → `com.docencia.(listas|composicion|herencia).ejercicio1..16`

Fundamentos

A continuación se describe qué se solicita en cada bloque de fundamentos.

1) Construcción de clases (`com.docencia.clases`)

En cada ejercicio hay una clase del dominio con:

- constructor vacío
- constructor con identificador único
- `equals`, `hashCode` y `toString`

- **Ejercicio 1**: implementar `equals/hashCode/toString` en `Persona` usando `dni` como identificador único.
- **Ejercicio 2**: implementar `equals/hashCode/toString` en `Libro` usando `isbn` como identificador único.
- **Ejercicio 3**: implementar `equals/hashCode/toString` en `Coche` usando `matricula` como identificador único.
- **Ejercicio 4**: implementar `equals/hashCode/toString` en `CuentaBancaria` usando `iban` como identificador único.
- **Ejercicio 5**: implementar `equals/hashCode/toString` en `Cliente` usando `id` como identificador único.
- **Ejercicio 6**: implementar `equals/hashCode/toString` en `Pedido` usando `codigo` como identificador único.
- **Ejercicio 7**: implementar `equals/hashCode/toString` en `Pelicula` usando `codigo` como identificador único.
- **Ejercicio 8**: implementar `equals/hashCode/toString` en `Usuario` usando `username` como identificador único.
- **Ejercicio 9**: implementar `equals/hashCode/toString` en `Curso` usando `codigo` como identificador único.
- **Ejercicio 10**: implementar `equals/hashCode/toString` en `Mascota` usando `chip` como identificador único.

> **IMPORTANTE**: puede parecer repetitivo (y lo es), pero `constructores/getters/setters/equals/hashCode/toString` es una base fundamental en **Java**.

2) Bucles `for` sobre arrays (`com.docencia.arrays`)

- **Ejercicio 1**: `sumar(int[] numeros)` → Suma todos los elementos del array. Si es `null` o vacío, devuelve 0.
- **Ejercicio 2**: `maximo(int[] numeros)` → Devuelve el valor máximo del array. Si es `null` o vacío, lanza `IllegalArgumentException`.
- **Ejercicio 3**: `contarPares(int[] numeros)` → Cuenta cuántos elementos del array son pares.
- **Ejercicio 4**: `invertir(int[] numeros)` → Devuelve un nuevo array con los elementos en orden inverso (sin modificar el original).
- **Ejercicio 5**: `indiceDe(int[] numeros, int objetivo)` → Devuelve el índice de `objetivo` o -1 si no está.
- **Ejercicio 6**: `mediaEntera(int[] numeros)` → Devuelve la media entera (suma/longitud). Si es `null` o vacío, devuelve 0.
- **Ejercicio 7**: `sonIguales(int[] a, int[] b)` → Devuelve `true` si ambos arrays tienen la misma longitud y mismos valores en cada posición.

- **Ejercicio 8**: `contarOcurrencias(int[] numeros, int valor)` → Cuenta cuántas veces aparece `valor` en el array.
- **Ejercicio 9**: `concatenarConGuion(String[] palabras)` → Devuelve un `String` uniendo con `-` usando `for`. Si es `null` o vacío, devuelve `""`.
- **Ejercicio 10**: `normalizarNegativos(int[] numeros)` → Devuelve un nuevo array donde los negativos se sustituyen por 0.

3) Condiciones y `switch` (`com.docencia.condiciones`)

- **Ejercicio 1**: `clasificarNota(int nota)` → Devuelve `"SUSPENSO"`, `"APROBADO"`, `"NOTABLE"` o `"SOBRESALIENTE"` según la nota (0–10).
- **Ejercicio 2**: `nombreDia(int dia)` → Usa `switch`: 1=Lunes ... 7=Domingo. Si no es válido devuelve `"ERROR"`.
- **Ejercicio 3**: `calculadora(int a, int b, char op)` → Usa `switch` con `+`, `-`, `*`, `/`. División entera. Si `op` no es válido lanza `IllegalArgumentException`.
- **Ejercicio 4**: `precioEntrada(int edad)` → if/else: <4 gratis, 4–17 5.0, 18–64 10.0, ≥65 6.0.
- **Ejercicio 5**: `esBisiesto(int anio)` → Devuelve `true` si el año es bisiesto.
- **Ejercicio 6**: `diasDelMes(int mes, int anio)` → Usa `switch` para devolver días del mes (1–12). Febrero depende de bisiesto.
- **Ejercicio 7**: `esVocal(char c)` → Devuelve `true` si `c` es vocal (mayúscula o minúscula).
- **Ejercicio 8**: `costeEnvio(double pesoKg, int zona)` → `switch` por zona: 1=5€/kg, 2=7€/kg, 3=10€/kg. Zona inválida lanza `IllegalArgumentException`.
- **Ejercicio 9**: `categoriaIMC(double imc)` → if/else: <18.5 `"BAJO"`, <25 `"NORMAL"`, <30 `"SOBREPESO"`, resto `"OBESIDAD"`.
- **Ejercicio 10**: `accionSemaforo(String color)` → `switch` sobre `"ROJO"`, `"AMARILLO"`, `"VERDE"`. Devuelve `"PARAR"`, `"PRECAUCION"`, `"AVANZAR"`. Si otro, `"ERROR"`.

4) Mixtos (`com.docencia.mixto`)

Cada ejercicio mixto combina:

- clase con id único (`equals/hashCode/toString`)
- recorrido de arrays con `for`
- decisiones con `if/else` y/o `switch`
- **Ejercicio 1**: trabajar con `Producto` (id `codigo`) y completar `Ejercicio1`.
- **Ejercicio 2**: trabajar con `Alumno` (id `dni`) y completar `Ejercicio2`.
- **Ejercicio 3**: trabajar con `Empleado` (id `nif`) y completar `Ejercicio3`.
- **Ejercicio 4**: trabajar con `Vehiculo` (id `matricula`) y completar `Ejercicio4`.
- **Ejercicio 5**: trabajar con `Reserva` (id `localizador`) y completar `Ejercicio5`.
- **Ejercicio 6**: trabajar con `Ticket` (id `id`) y completar `Ejercicio6`.
- **Ejercicio 7**: trabajar con `Libro` (id `isbn`) y completar `Ejercicio7`.

- **Ejercicio 8**: trabajar con `Pedido` (id `codigo`) y completar `Ejercicio8`.
- **Ejercicio 9**: trabajar con `LecturaSensor` (id `sensorId`) y completar `Ejercicio9`.
- **Ejercicio 10**: trabajar con `Movimiento` (id `codigo`) y completar `Ejercicio10`.
- **Ejercicio 11**: trabajar con `ClaseAula` (id `codigo`) y completar `Ejercicio11`.
- **Ejercicio 12**: trabajar con `Envio` (id `tracking`) y completar `Ejercicio12`.
- **Ejercicio 13**: trabajar con `Partido` (id `id`) y completar `Ejercicio13`.
- **Ejercicio 14**: trabajar con `Articulo` (id `sku`) y completar `Ejercicio14`.
- **Ejercicio 15**: trabajar con `Alumno2` (id `id`) y completar `Ejercicio15`.
- **Ejercicio 16**: trabajar con `Alumno3` (id `dni`) y completar `Ejercicio16`.
- **Ejercicio 17**: trabajar con `Factura` (id `numero`) y completar `Ejercicio17`.
- **Ejercicio 18**: trabajar con `Juego` (id `codigo`) y completar `Ejercicio18`.
- **Ejercicio 19**: trabajar con `Ruta` (id `id`) y completar `Ejercicio19`.
- **Ejercicio 20**: trabajar con `Examen` (id `codigo`) y completar `Ejercicio20`.

> **Importante**: recuerda que las clases suelen estar sin completar (constructores, `equals`, etc.).

Listas + Composición + Herencia (simulacros, bloque unificado)

En este bloque encontrarás ejercicios de:

- **Listas**: tratamiento de `List<String>`, `List<Integer>`, null-safety, filtros, acumulaciones.
- **Composición**: modelado con clases relacionadas, colecciones internas, reglas de negocio.
- **Herencia**: polimorfismo, clases base/derivadas, comportamiento común.

Estructura

La estructura de paquetes es homogénea:

```
com.docencia.(listas|composicion|herencia).ejercicioN
```

En cada paquete encontrarás:

- `EjercicioN.java` (con `main`) para pruebas manuales.
- Las clases de apoyo del ejercicio en ficheros **independientes** dentro del mismo paquete (si aplica).
- `EjercicioNTest.java` con los tests.

> A continuación, cada **número de ejercicio** agrupa: **Listas**, **Composición** y **Herencia** con el mismo índice `N`.

Ejercicio 1

Listas — Contar mayores que un umbral (`com.docencia.listas.ejercicio1.Ejercicio1`)

- Clase: `com.docencia.listas.ejercicio1.Ejercicio1`
- Método:

```
public static int contarMayoresQue(java.util.List<Integer> numeros,
int umbral)
```

- Enunciado:
  Dada una lista de enteros, contar cuántos valores son **estrictamente mayores** que el umbral indicado.
  Debes ignorar elementos `null` y tratar correctamente listas `null` o vacías.

Composición — Gestor de tareas (`com.docencia.composicion.ejercicio1.Ejercicio1`)

- Clases:
  - `Tarea` — descripción y estado (completada o no).
  - `ListaTareas` — mantiene una lista de tareas.
- Funcionalidad:
  - `void anadirTarea(String descripcion)` → añade una nueva tarea pendiente si la descripción es válida (no `null` ni en blanco).
  - `boolean marcarComoCompletada(String descripcion)` → marca como completada la **primera** tarea cuya descripción coincida (case-insensitive, con `trim()`).
  - `java.util.List<Tarea> obtenerPendientes()` → devuelve una **lista nueva** con todas las tareas no completadas.

Herencia — Figuras geométricas (`com.docencia.herencia.ejercicio1.Ejercicio1`)

- Jerarquía:
  - `Figura` (abstracta) → `double area()`
  - `Circulo` (radio)
  - `Rectangulo` (ancho y alto)
- Funcionalidad:
  - Implementar `area()` en cada subtipo.
  - Método estático:

```
public static double areaTotal(java.util.List<Figura> figuras)
```

    suma el área de todas las figuras no nulas (si la lista es `null` o no hay válidas, devuelve `0.0`).

Ejercicio 2

Listas — Invertir y filtrar cadenas (`com.docencia.listas.ejercicio2.Ejercicio2`)

- Clase: `com.docencia.listas.ejercicio2.Ejercicio2`
- Método:

```
public static java.util.List<String>
invertirYFiltrarNoVacias(java.util.List<String> cadenas)
```

- Enunciado:
  Construir una **nueva lista** con la versión invertida de cada cadena **no nula** y **no vacía/en blanco**, respetando el orden original.
  Las cadenas `null` o en blanco se descartan.

Composición — Notas de alumnos (`com.docencia.composicion.ejercicio2.Ejercicio2`)

- Clases:
  - `Alumno` — nombre.
  - `RegistroNota` — alumno, asignatura, nota numérica.
  - `Aula` — lista de registros de notas.
- Funcionalidad:
  - `void registrarNota(String nombreAlumno, String asignatura, double nota)` → registra si nombre/asignatura son válidos y `nota` está en `[0,10]`.
  - `double mediaAlumno(String nombreAlumno)` → media de notas del alumno (nombre case-insensitive). Si no hay notas válidas, `0.0`.

Herencia — Empleados y nómina (`com.docencia.herencia.ejercicio2.Ejercicio2`)

- Jerarquía:
  - `Empleado` (abstracta) → `double calcularSalarioMensual()`
  - `EmpleadoFijo` (salario mensual)
  - `EmpleadoPorHoras` (horas y tarifa)
- Funcionalidad:
  - Método estático:

```
public static double costeTotalNomina(java.util.List<Empleado>
empleados)
```

    suma salarios de empleados no nulos (si lista `null`/vacía → `0.0`).

Ejercicio 3

Listas — Intersección ordenada (`com.docencia.listas.ejercicio3.Ejercicio3`)

- Clase: `com.docencia.listas.ejercicio3.Ejercicio3`
- Método:

```
public static java.util.List<Integer>
interseccionOrdenada(java.util.List<Integer> a,
java.util.List<Integer> b)
```

- Enunciado:
  Obtener la lista de elementos que aparecen en **ambas listas** (intersección), **sin duplicados**, ordenada de menor a mayor.
  Ignora `null` y trata listas `null` como listas vacías.

Composición — Centro de salud y consultas (`com.docencia.composicion.ejercicio3.Ejercicio3`)

- Clases:
  - `Paciente` — nombre.
  - `Consulta` — paciente y motivo.
  - `CentroSalud` — lista de consultas.
- Funcionalidad:
  - `void registrarConsulta(Paciente paciente, String motivo)` → registra si paciente no es `null`, nombre válido y motivo válido.
  - `int contarConsultasDe(String nombrePaciente)` → cuenta consultas de un paciente (case-insensitive y con `trim()`); si nombre inválido → `0`.

Herencia — Animales y sonidos (`com.docencia.herencia.ejercicio3.Ejercicio3`)

- Jerarquía:
  - `Animal` (abstracta) → `String sonido()`
  - `Perro` → `"guau"`
  - `Gato` → `"miau"`
- Funcionalidad:
  - Método estático:

```
public static String concatenarSonidos(java.util.List<Animal>
animales)
```

    concatena sonidos de animales no nulos separados por un espacio (si no hay válidos → `""`).

Ejercicio 4

Listas — Media de valores válidos (`com.docencia.listas.ejercicio4.Ejercicio4`)

- Clase: `com.docencia.listas.ejercicio4.Ejercicio4`

- Método:

```
public static double mediaValoresValidos(java.util.List<Integer>
valores)
```

- Enunciado:
  Calcular la media aritmética de los valores no nulos.
  Si la lista es `null`, vacía, o todos sus elementos son `null`, la media debe ser `0.0`.

Composición — Mensajería interna (`com.docencia.composicion.ejercicio4.Ejercicio4`)

- Clases:
  - `Usuario` — nombre.
  - `Mensaje` — remitente, destinatario y texto.
  - `BandejaMensajes` — lista de mensajes.
- Funcionalidad:
  - `void enviarMensaje(String remitente, String destinatario, String texto)` → registra solo si los tres campos son válidos.
  - `java.util.List<Mensaje> mensajesPara(String destinatario)` → devuelve una lista nueva con mensajes para ese destinatario (case-insensitive); si destinatario inválido → lista vacía.

Herencia — Publicaciones recientes (`com.docencia.herencia.ejercicio4.Ejercicio4`)

- Jerarquía:
  - `Publicacion` (abstracta) — título y año.
  - `LibroPublicacion` — reciente si ≤ 5 años.
  - `ArticuloPublicacion` — reciente si ≤ 2 años.
- Funcionalidad:
  - Método en subtipos:

    ```
    public abstract boolean esReciente(int anioActual);
    ```

  - Método estático:

    ```
    public static int contarRecientes(java.util.List<Publicacion>
    publicaciones, int anioActual)
    ```

    cuenta publicaciones recientes (si lista `null` → `0`).

Ejercicio 5

Listas — Normalizar nombres (`com.docencia.listas.ejercicio5.Ejercicio5`)

- Clase: `com.docencia.listas.ejercicio5.Ejercicio5`
- Método:

```
public static java.util.List<String>
normalizarNombres(java.util.List<String> nombres)
```

- Enunciado:
  Devolver una nueva lista con nombres normalizados:
  - ignorar `null` o en blanco
  - hacer `trim()`
  - primera letra en mayúscula y resto en minúscula
    Conserva el orden.

Composición — Agenda de contactos (`com.docencia.composicion.ejercicio5.Ejercicio5`)

- Clases:
  - `Contacto` — nombre y teléfono.
  - `AgendaContactos` — lista de contactos.
- Funcionalidad:
  - `void anadirContacto(String nombre, String telefono)` → añade si nombre válido y teléfono tiene **9 dígitos**.
  - `String buscarTelefono(String nombre)` → devuelve teléfono del primer contacto cuyo nombre coincida (case-insensitive, con `trim()`); si no existe → `null`.

Herencia — Cuentas bancarias (`com.docencia.herencia.ejercicio5.Ejercicio5`)

- Jerarquía:
  - `CuentaBancaria` (abstracta) — número y saldo; `ingresar`, `retirar`.
  - `CuentaCorriente` — comisión fija al retirar.
  - `CuentaAhorro` — no permite retiradas que dejen saldo negativo.
- Funcionalidad:
  - `ingresar(double cantidad)` → solo cantidades positivas.
  - `retirar(double cantidad)` → reglas según tipo (ver tests).
  - Método estático `saldoTotal(List<CuentaBancaria> cuentas)` → suma saldos (si lista `null` → `0.0`).

Ejercicio 6

Listas — Contar ocurrencias de un texto (`com.docencia.listas.ejercicio6.Ejercicio6`)

- Clase: `com.docencia.listas.ejercicio6.Ejercicio6`
- Método:

```
public static int contarOcurrencias(java.util.List<String> textos,
```

```
String objetivo)
```

- Enunciado:
  Contar coincidencias con `objetivo` ignorando mayúsculas/minúsculas y espacios.
  Si lista `null` o objetivo inválido → `0`.

Composición — Carrito de la compra (`com.docencia.composicion.ejercicio6.Ejercicio6`)

- Clases:
  - `Producto` — nombre y precio.
  - `LineaCarrito` — producto y cantidad.
  - `CarritoCompra` — lista de líneas.
- Funcionalidad:
  - `void anadirLinea(Producto producto, int cantidad)` → solo si producto válido, precio ≥ 0 y cantidad > 0.
  - `double calcularTotal(double porcentajeIva)` → total = sumaBase * (1 + iva/100).

Herencia — Dispositivos electrónicos (`com.docencia.herencia.ejercicio6.Ejercicio6`)

- Jerarquía:
  - `Dispositivo` (abstracta) — `double consumoHora()`
  - `Bombilla` — vatios (W)
  - `Calefactor` — potencia (W) y modo eco (reduce consumo)
- Funcionalidad:
  - Método estático `consumoTotalHora(List<Dispositivo> dispositivos)` → suma consumos de no nulos.

Ejercicio 7

Listas — Concatenar con separador (`com.docencia.listas.ejercicio7.Ejercicio7`)

- Clase: `com.docencia.listas.ejercicio7.Ejercicio7`
- Método:

```
public static String concatenarConSeparador(java.util.List<String>
partes, String separador)
```

- Enunciado:
  Concatenar partes **válidas** (no `null` ni en blanco) usando el separador.
  Si `separador` es `null`, usar `", "`. Si no hay partes válidas → `""`.

Composición — Gestor de incidencias (`com.docencia.composicion.ejercicio7.Ejercicio7`)

- Clases:
  - `Incidencia` — id, descripción y estado (abierta/cerrada).
  - `GestorIncidencias` — colección de incidencias.

- Funcionalidad:
  - `Incidencia crearIncidencia(String descripcion)` → crea con id incremental (1,2,3…) y estado **ABIERTA**; si descripción inválida → `null`.
  - `boolean cerrarIncidencia(int id)` → cierra si existe abierta con ese id.
  - `long contarAbiertas()` → devuelve cuántas están abiertas.

Herencia — Personas y roles (`com.docencia.herencia.ejercicio7.Ejercicio7`)

- Jerarquía:
  - `Persona` (abstracta) — `String descripcionRol()`
  - `Estudiante` — añade curso
  - `Profesor` — añade especialidad
- Funcionalidad:
  - Método estático `java.util.List<String> descripciones(List<Persona> personas)` → devuelve descripciones en orden, ignorando `null`.

Ejercicio 8

Listas — Máximo seguro (`com.docencia.listas.ejercicio8.Ejercicio8`)

- Clase: `com.docencia.listas.ejercicio8.Ejercicio8`
- Método:

```
public static Integer maximoSeguro(java.util.List<Integer> numeros)
```

- Enunciado:
  Devolver el máximo de los elementos no nulos.
  Si la lista es `null`, vacía, o todos son `null` → devolver `null`.

Composición — Registro de asistencia (`com.docencia.composicion.ejercicio8.Ejercicio8`)

- Clases:
  - `Alumno` — nombre.
  - `RegistroAsistencia` — alumno, día y presente/ausente.
  - `RegistroClase` — colección de registros.
- Funcionalidad:
  - `void registrarAsistencia(String nombreAlumno, String dia, boolean presente)` → registra solo si nombre y día son válidos.
  - `double porcentajeAsistencia(String nombreAlumno)` → porcentaje asistido; si no hay registros → `0.0`.

Herencia — Envíos y urgencia (`com.docencia.herencia.ejercicio8.Ejercicio8`)

- Jerarquía:
  - `Envio` (abstracta) — destino y peso (kg) → `double calcularCoste()`
  - `EnvioNacional`

    - `EnvioInternacional`
- Funcionalidad:
    - Método estático `costeTotal(List<Envio> envios)` → suma costes de envíos no nulos (si lista `null`/vacía → `0.0`).

Ejercicio 9

Listas — Sumar positivos (`com.docencia.listas.ejercicio9.Ejercicio9`)

- Clase: `com.docencia.listas.ejercicio9.Ejercicio9`
- Método:

```
public static int sumarPositivos(java.util.List<Integer> numeros)
```

- Enunciado:
  Sumar valores **estrictamente positivos** ignorando `null` y valores ≤ 0.
  Si lista `null`/vacía → `0`.

Composición — Inventario de productos (`com.docencia.composicion.ejercicio9.Ejercicio9`)

- Clases:
    - `Producto` — nombre.
    - `LineaInventario` — producto y cantidad.
    - `Inventario` — lista de líneas.
- Funcionalidad:
    - `void anadirStock(String nombreProducto, int cantidad)` → solo si nombre válido y cantidad > 0; si existe producto (nombre case-insensitive + `trim`) suma stock, si no crea nueva línea.
    - `boolean retirarStock(String nombreProducto, int cantidad)` → retira si hay stock suficiente y datos válidos.
    - `int stockDe(String nombreProducto)` → stock del producto (si nombre inválido/no existe → `0`).
    - `int totalUnidades()` → suma de cantidades de todas las líneas.

Herencia — Pagos (`com.docencia.herencia.ejercicio9.Ejercicio9`)

- Jerarquía:
    - `Pago` (abstracta) — importe base → `double calcularImporteFinal()`
    - `PagoEfectivo` — importe final = `max(0, importeBase)`
    - `PagoTarjeta` — recargo porcentual (si base negativa se toma 0)
- Funcionalidad:
    - Método estático `totalPagos(List<Pago> pagos)` → suma importes finales (si lista `null` → `0.0`).

Ejercicio 10

Listas — Filtrar por prefijo (`com.docencia.listas.ejercicio10.Ejercicio10`)

- Clase: `com.docencia.listas.ejercicio10.Ejercicio10`
- Método:

```
public static java.util.List<String>
filtrarPorPrefijo(java.util.List<String> textos, String prefijo)
```

- Enunciado:
  Devolver una nueva lista con textos que empiezan por el prefijo (case-insensitive y con `trim`), descartando `null`/blancos.
  Si prefijo inválido → lista vacía.

Composición — Gestor de proyectos y tareas (`com.docencia.composicion.ejercicio10.Ejercicio10`)

- Clases:
  - `Tarea` — descripción y completada.
  - `Proyecto` — nombre y lista de tareas.
  - `GestorProyectos` — lista de proyectos.
- Funcionalidad:
  - `Proyecto crearProyecto(String nombre)` → crea y añade si nombre válido; si no, `null`.
  - `boolean anadirTareaAProyecto(String nombreProyecto, String descripcionTarea)` → añade tarea si proyecto existe (nombre case-insensitive + `trim`) y descripción válida.
  - `int contarTareasPendientes(String nombreProyecto)` → cuenta pendientes; si proyecto no existe → `0`.

Herencia — Contenidos multimedia (`com.docencia.herencia.ejercicio10.Ejercicio10`)

- Jerarquía:
  - `Contenido` (abstracta) — título y duración (min) → `boolean esLargo()`
  - `Pelicula` — largo si ≥ 90 min
  - `Podcast` — largo si ≥ 60 min
- Funcionalidad:
  - Método estático `contarLargos(List<Contenido> contenidos)` → cuenta largos ignorando `null` (lista `null` → `0`).

Ejercicio 11

Listas — Ordenar descendente sin `null` (`com.docencia.listas.ejercicio11.Ejercicio11`)

- Clase: `com.docencia.listas.ejercicio11.Ejercicio11`
- Método:

```
public static java.util.List<Integer>
ordenarDescSinNulls(java.util.List<Integer> numeros)
```

- Enunciado:
  Devolver una nueva lista con valores no nulos ordenados de mayor a menor.
  Si lista `null` → lista vacía.

Composición — Club de lectura (`com.docencia.composicion.ejercicio11.Ejercicio11`)

- Clases:
  - `Socio` — nombre.
  - `Lectura` — socio y título.
  - `ClubLectura` — lista de lecturas.
- Funcionalidad:
  - `void registrarLectura(String nombreSocio, String tituloLibro)` → registra si ambos válidos.
  - `int vecesLeido(String tituloLibro)` → cuenta lecturas de un título (case-insensitive + `trim`); si inválido → `0`.
  - `String libroMasLeido()` → devuelve el título con más lecturas; empate: el que llegue antes al máximo; si no hay lecturas → `null`.

Herencia — Notificaciones (`com.docencia.herencia.ejercicio11.Ejercicio11`)

- Jerarquía:
  - `Notificacion` (abstracta) — mensaje → `String formatear()`
  - `NotificacionEmail` — destinatario y asunto
  - `NotificacionSms` — teléfono
- Funcionalidad:
  - Método estático `formatearTodas(List<Notificacion> notificaciones)` → devuelve lista de textos formateados (ignora `null`; lista `null` → lista vacía).

Ejercicio 12

Listas — Unión ordenada sin duplicados (`com.docencia.listas.ejercicio12.Ejercicio12`)

- Clase: `com.docencia.listas.ejercicio12.Ejercicio12`
- Método:

```
public static java.util.List<Integer>
unionOrdenadaSinDuplicados(java.util.List<Integer> a,
java.util.List<Integer> b)
```

- Enunciado:
  Construir una lista con valores no nulos que aparezcan en `a` o `b`, sin duplicados y ordenados de

menor a mayor.
Listas `null` se tratan como vacías.

Composición — Sistema de votos (`com.docencia.composicion.ejercicio12.Ejercicio12`)

- Clases:
  - `Candidato` — nombre.
  - `Voto` — candidato.
  - `MesaElectoral` — lista de votos.
- Funcionalidad:
  - `void registrarVoto(String nombreCandidato)` → registra voto si nombre válido.
  - `int votosDe(String nombreCandidato)` → cuenta votos (case-insensitive + `trim`); si inválido → `0`.
  - `String ganador()` → candidato con más votos; empate: quien alcanzó antes el máximo; sin votos → `null`.

Herencia — Figuras 3D (`com.docencia.herencia.ejercicio12.Ejercicio12`)

- Jerarquía:
  - `Figura3D` (abstracta) — `double volumen()`
  - `Cubo` — lado
  - `Esfera` — radio
- Funcionalidad:
  - Método estático `volumenTotal(List<Figura3D> figuras)` → suma volúmenes (si lista `null`/vacía → `0.0`).

Ejercicio 13

Listas — Filtrar mayores y ordenar (`com.docencia.listas.ejercicio13.Ejercicio13`)

- Clase: `com.docencia.listas.ejercicio13.Ejercicio13`
- Método:

```
public static java.util.List<Integer>
filtrarMayoresYOrdenar(java.util.List<Integer> numeros, int minimo)
```

- Enunciado:
  Devolver una nueva lista con valores no nulos **≥ mínimo**, ordenados de menor a mayor.
  Si lista `null` o sin válidos → lista vacía.

Composición — Biblioteca simple (`com.docencia.composicion.ejercicio13.Ejercicio13`)

- Clases:
  - `Libro` — título, autor, año.
  - `Biblioteca` — lista de libros.
- Funcionalidad:

  - `void agregarLibro(String titulo, String autor, int anio)` → agrega si título/autor válidos y `anio > 0`.
  - `java.util.List<Libro> buscarPorAutor(String autor)` → devuelve una lista **copia** con coincidencias (autor case-insensitive + `trim`).
  - `long contarLibrosAnterioresA(int anioLimite)` → cuenta libros con año **< anioLimite**.

Herencia — Vehículos y velocidad (`com.docencia.herencia.ejercicio13.Ejercicio13`)

- Jerarquía:
  - `Vehiculo` (abstracta) — marca y velocidad máxima → `boolean esRapido()`
  - `Coche` — puertas (rápido si velocidad ≥ 180)
  - `Moto` — cilindrada (rápida si velocidad ≥ 140)
- Funcionalidad:
  - Método estático `contarRapidos(List<Vehiculo> vehiculos)` → cuenta rápidos ignorando `null` (lista `null` → `0`).

Ejercicio 14

Listas — Eliminar duplicados preservando orden (`com.docencia.listas.ejercicio14.Ejercicio14`)

- Clase: `com.docencia.listas.ejercicio14.Ejercicio14`
- Método:

```
public static java.util.List<String>
eliminarDuplicadosPreservandoOrden(java.util.List<String> textos)
```

- Enunciado:
  Devolver una nueva lista sin `null`, sin duplicados (comparación **case-sensitive**) y conservando el orden de primera aparición.
  Si lista `null` → lista vacía.

Composición — Restaurante y pedidos (`com.docencia.composicion.ejercicio14.Ejercicio14`)

- Clases:
  - `Plato` — nombre y precio.
  - `Pedido` — plato y cantidad.
  - `MesaPedidos` — lista de pedidos.
- Funcionalidad:
  - `void anadirPedido(Plato plato, int cantidad)` → añade si plato válido, precio ≥ 0 y cantidad > 0.
  - `double importeTotal()` → suma `precio * cantidad`.
  - `double importeConDescuento(double porcentaje)` → aplica descuento (`porcentaje` negativo se trata como 0).

Herencia — Documentos y descripción (`com.docencia.herencia.ejercicio14.Ejercicio14`)

- Jerarquía:
  - `Documento` (abstracta) — título → `String descripcion()`
  - `Informe` — páginas
  - `Carta` — destinatario
- Funcionalidad:
  - Método estático `descripciones(List<Documento> documentos)` → devuelve lista de descripciones (ignora `null`; lista `null` → lista vacía).

Ejercicio 15

Listas — Buscar primera cadena que contenga (`com.docencia.listas.ejercicio15.Ejercicio15`)

- Clase: `com.docencia.listas.ejercicio15.Ejercicio15`
- Método:

```
public static String buscarPrimeraQueContiene(java.util.List<String> textos, String fragmento)
```

- Enunciado:
  Devolver el **primer** texto que contenga el fragmento (case-insensitive, con `trim` en ambos).
  Si lista `null`, fragmento inválido o no hay coincidencias → `null`.

Composición — Empresa y salarios (`com.docencia.composicion.ejercicio15.Ejercicio15`)

- Clases:
  - `Trabajador` — nombre y salario.
  - `Departamento` — nombre y lista de trabajadores.
  - `Empresa` — lista de departamentos.
- Funcionalidad:
  - `Departamento crearDepartamento(String nombre)` → crea y añade si nombre válido; si no, `null`.
  - `boolean anadirTrabajador(String nombreDepartamento, String nombreTrabajador, double salario)` → añade si existe departamento (nombre case-insensitive + `trim`), trabajador válido y salario > 0.
  - `double salarioMedioDepartamento(String nombreDepartamento)` → media; si no existe o sin trabajadores → `0.0`.
  - `double salarioTotalEmpresa()` → suma salarios de toda la empresa.

Herencia — Cursos y carga lectiva (`com.docencia.herencia.ejercicio15.Ejercicio15`)

- Jerarquía:
  - `Curso` (abstracta) — nombre y horas → `boolean esIntenso()`
  - `CursoOnline` — plataforma

    - `CursoPresencial` — aula
- Funcionalidad:
    - Se considera intenso si horas ≥ 40.
    - Método estático `contarIntensos(List<Curso> cursos)` → cuenta intensos ignorando `null`.

Ejercicio 16

Listas — Sumar longitudes de cadenas no vacías (`com.docencia.listas.ejercicio16.Ejercicio16`)

- Clase: `com.docencia.listas.ejercicio16.Ejercicio16`
- Método:

```
public static int sumarLongitudesNoVacias(java.util.List<String> textos)
```

- Enunciado:
  Sumar la longitud de todas las cadenas **no nulas y no en blanco**.
  Si lista `null` o sin cadenas válidas → `0`.

Composición — Registro de temperaturas (`com.docencia.composicion.ejercicio16.Ejercicio16`)

- Clases:
    - `Medicion` — día (`String`) y temperatura (`double`).
    - `EstacionMeteorologica` — lista de mediciones.
- Funcionalidad:
    - `void registrarMedicion(String dia, double temperatura)` → registra si el día es válido.
    - `Double maximaTemperatura()` → devuelve la máxima o `null` si no hay mediciones.
    - `long diasPorEncimaDe(double umbral)` → cuenta mediciones con temperatura **> umbral**.

Herencia — Ofertas y precio final (`com.docencia.herencia.ejercicio16.Ejercicio16`)

- Jerarquía:
    - `Oferta` (abstracta) — precio base → `double precioFinal()`
    - `OfertaPorcentaje` — descuento porcentual
    - `OfertaCantidadFija` — descuento fijo
- Funcionalidad:
    - El precio final **nunca puede ser negativo**.
    - Método estático `sumaPreciosFinales(List<Oferta> ofertas)` → suma precios finales (lista `null` → `0.0`).