# Gestión de Inscritos a un Taller

Eres responsable de administrar la lista de **participantes** de un taller de programación. Necesitas un pequeño programa en Java que permita **registrar, actualizar, buscar y eliminar** participantes, controlando **cupos** y **datos válidos**. El objetivo es que todas las operaciones sobre la lista estén encapsuladas en un **TallerManager**, y que puedas **corroborar** los cambios mostrando el estado antes y después.

## Objetivos de aprendizaje

- Usar **ArrayList** para almacenar y manipular objetos.
- Aplicar **composición**: `Taller` contiene una `ArrayList<Participante>`.
- Centralizar reglas en un **Manager** (validaciones + operaciones).
- Practicar métodos típicos de `ArrayList`: `add`, `get`, `set`, `remove`, `size`, `isEmpty`, `clear`, `contains` (opcional).
- Diseñar **métodos de servicio** con precondiciones y mensajes de resultado.

---

## Modelo de datos (clases)

### 1) `Participante`

**Atributos (privados):**

- `String rut` (o ID único)
- `String nombre`
- `String email`
- `boolean activo` (true = inscrito/participando; false = inactivo)

**Constructores:**

- Completo con todos los campos.

**Métodos (públicos mínimos):**

- Getters/Setters.
- `String toString()` legible (incluye rut, nombre, email, estado).
- *(Opcional)* `boolean equals(Object o)` y `int hashCode()` basados en `rut` para poder usar `contains/remove(Object)` fácilmente.

## 2) `Taller`

**Atributos (privados):**

- `String nombre`
- `int cupoMaximo`
- `ArrayList<Participante> participantes`

**Constructores:**

- Con `nombre` y `cupoMaximo` (inicializa `participantes` vacío).

**Métodos (públicos mínimos):**

- Getters de `nombre`, `cupoMaximo`, `participantes` (si devuelves la lista, que sea **sólo lectura** en la práctica: úsala para imprimir; las mutaciones deben ir por el Manager).

## 3) `TallerManager`

**Atributos (privados):**

- `Taller taller`

**Constructor:**

- Recibe un `Taller`.

**Métodos (públicos):**

1. `boolean registrar(Participante p)`
   - **Validaciones**:
     - No `null`.
     - `rut` único (no repetido en la lista).
     - `email` con formato simple: contiene `'@'` y `'.'`.
     - No superar `cupoMaximo`.
   - **Efecto**: agrega a la lista si es válido; retorna `true/false` y muestra mensaje.
2. `boolean actualizarEmail(String rut, String nuevoEmail)`
   - **Validación**: formato de email; existencia del `rut`.
   - **Efecto**: hace `set` del email y confirma el cambio.
3. `boolean cambiarEstado(String rut, boolean activo)`
   - Cambia `activo` ⇄ `inactivo` para el participante indicado.
4. `boolean eliminarPorRut(String rut)`

- Elimina al participante por índice o por objeto.
- Retorna si efectivamente se eliminó.

5. `ArrayList<Participante> buscarPorNombre(String fragmento)`
   - Retorna **nueva** lista con coincidencias *case-insensitive* usando `indexOf`/`toLowerCase`.
6. `Participante buscarPorRut(String rut)`
   - Retorna el objeto o `null`.

**Métodos (privados de apoyo):**

- `int indicePorRut(String rut)` → `-1` si no existe.

---

# Casos de uso requeridos (demostrables en `main`)

Implementa un `main` que:

1. **Carga inicial**
   - Crea un `Taller("Taller Java", 5)` y un `TallerManager`.
   - Inserta **3–5 participantes** válidos con `registrar(...)`.
   - Muestra: "**Estado inicial**" → cantidad (`size`) y lista (`toString` por línea).
2. **Registro con validaciones**
   - Intenta registrar:
     - a) un `rut` duplicado (debe fallar).
     - b) un email inválido (debe fallar).
     - c) registros hasta **llenar el cupo** (el último exceso debe fallar).
   - Muestra mensajes y **conteo final** tras los intentos.
3. **Actualización**
   - Elige un `rut` existente y cambia su email a uno válido.
   - Muestra **antes** y **después** del objeto (o al menos del campo).
4. **Cambio de estado**
   - Marca inactivo a un participante y verifica en la impresión.
5. **Búsqueda**
   - Ejecuta `buscarPorNombre("an")` (o fragmento que tenga resultados) y muestra la **sublista** devuelta.
6. **Eliminación**
   - Elimina por `rut` un participante existente.
   - Muestra "**Estado final**" (size y elementos).

> Cada bloque debe imprimir un encabezado y salidas claras para **corroborar** que la lista cambió.

---

## Reglas de validación (resumen rápido)

- `rut`: no nulo/ vacío y único.
- `email`: contiene `'@'` y `'.'`, sin espacios.
- **Cupo:** `participantes.size() < cupoMaximo` antes de `add`.
- Búsquedas y actualizaciones deben manejar “no existe” con mensajes legibles.

---

## Sugerencia de distribución de tiempo (40 min)

- 10 min: crear clases `Participante` y `Taller`.
- 15 min: `TallerManager` con `registrar`, `buscarPorRut`, `indicePorRut`.
- 10 min: `actualizarEmail`, `cambiarEstado`, `eliminarPorRut`, `buscarPorNombre`.
- 5 min: `main` de demostración con impresiones antes/después.