Algoritmos y Programación 1
Apellido y Nombres:
DNI:

E1	E2	E3	E4	E5	Calificación

Ejercicio 1 { java, arrays }

Implementar según la siguiente especificación, agregando pre y post-condiciones. Incorporar alguna validación de dominio.

Implementar la clase **ColeccionDeManga** tal que

1. Se construya a partir de la cantidad de tomos que tiene.
2. Agregue un tomo, indicando nombre y la cantidad de páginas que contiene. Como toda colección de Manga, se agrega de derecha a izquierda.
3. Devuelva la cantidad de páginas de un tomo dado su nombre.
4. Cuente la cantidad de tomos en la colección.
5. Calcule el tiempo que se tardará en llegar al tomo en la posición `x`, asumiendo que se lee una página por minuto y debo leer todos los tomos a la derecha del que deseo leer, antes de llegar a dicho tomo. ¡No quiero perderme ni un capítulo!

```
class ColeccionDeManga {
      public ColeccionDeManga(int) { /* ... */ }
      public void agregarTomo(String, int) { /* ... */ }
      public int obtenerPaginasParaTomo(String) { /* ... */ }
      public int cantidadDeTomos() { /* ... */ }
      public int tiempoParaLlegarATomo(int) { /* ... */ }
}
```

Ejercicio 2 { pruebas }

Plantear cinco (5) pruebas para el problema del punto anterior. Los casos planteados deberán ser **conceptualmente** diferentes. Escribir el código en JUnit para tres (3) de esos casos.

Algoritmos y Programación 1
Apellido y Nombres:
DNI:

Ejercicio 3 { memoria }

Realizar los diagramas de Stack y Heap para los momentos #uno y #dos del siguiente fragmento de código.

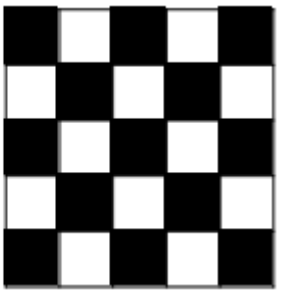
```
Grupo grupo = new Grupo(4);
Equipo argentina = new Equipo("Grecia");
Equipo grecia = new Equipo("Argentina");
Equipo noruega = grecia;
grecia = new Equipo("Uruguay");
// #uno
grecia = null;
grupo.agregar(grecia, 4);
grupo.agregar(argentina, 3);
grupo.agregar(new Equipo("Nigeria"), 1);
Assert.assertNull(grupo.obtenerEquipo(2));
Assert.assertNotNull(grupo.obtenerEquipo(4));
// #dos
```

Ejercicio 4 { algoritmos, matrices }

Dada una matriz de **Color** y de tamaño **n x n**, siendo **n** impar y mayor o igual a 3, y un **Color** determinado, escribir un algoritmo para pintar un lienzo que originalmente es de **Color.BLANCO** con el patrón solicitado. Se debe **pintar sobre el lienzo original**. No se recomienda pintar de **Color.BLANCO.**

```
public void pintarLienzo(Color[][] lienzo, Color pincel) {
    // tu código aquí
}
```

Se proporcionan ejemplos para n = 3 y n = 5



Ejercicio 5 { lógica }

Explique y ejemplifique con código Java el concepto de negación binaria (NOT). Escriba la tabla de verdad para dicho concepto. El ejemplo deberá ser aplicado y no conceptual (las variables deben tener sentido).