# Reporte de la Práctica 3: Conjuntos.

Julio César Lozano Garnica. No. de cuenta UNAM: 420095390

09 de septiembre del 2025

Este es el reporte de la tercera práctica de laboratorio de la materia *Estructuras de Datos* impartida por el profesor Erick Quintero Villeda y su ayudante de laboratorio Sandra Valeria Rivera Lara.

## Complicaciones al realizar la práctica.

En comparación con las dos prácticas anteriores, esta fue la que más trabajo me ha costado. Había muchas cosas que no sabía sobre los genéricos, la interfaz Iterable, en especial en la segunda sección de la práctica, en donde se resuelven los problemas con los .txt batallé mucho para conseguir separar las palabras de manera adecuada, al igual que el conseguir la intersección de los conjuntos. Al final, entendí que mi método agregarElemento, no estaba regresando nada y eso era el problema. Cambié la firma tanto en la interfaz Conjunto como en la clase ConjuntoArreglo para poder regresar los conjuntos que necesitaba y así completar la práctica.

## Definición formal del algoritmo del método interseccion e iguales.

---

**ALGORITMO 1.1** *interseccion*

---

**Entrada:** Un conjunto $c$ de tipo ConjuntoArreglo y un conjunto $c_2$ de tipo Conjunto.

**Salida:** Un conjunto $c_3$ de tipo Conjunto que resulta ser la intersección de $c$ con $c_2$.

1. Sea elementosInterAux un arreglo de longitud $[Math.min(this.obtenerCardinalidad(c), c.obtenerCardinalidad(c_2))]$
2. Sea contador = 0
3. Para cada elemento aux en c hacer:
4. Si aux es un elemento de $c$ hacer:
5. elementoInterAux[contador] = aux
6. contador++
7. end if
8. Sea elementosInter un arreglo de longitud contador
9. Para todo $o \leq$ i <contador hacer:
10. elementosInter[i] = elementosInterAux[i]

11. return new ConjuntoArrerglo<>(elementosInter)

---

**ALGORITMO 1.2** *iguales*

---

**Entrada:** Un conjunto $c$ de tipo ConjuntoArreglo y un conjunto $c_2$ de tipo Conjunto.

**Salida:** Verdadero si $c$ y $c_2$ tienen los mismos elementos, falso en otro caso.

1. return $c.contieneConjunto(c_2)$ && $c_2.contieneConjunto(c)$

---

## Parte de la práctica correspondiente a la definición de un TDA

La clase abstracta Conjunto<T>sería la definición del TDA Conjunto, pues se definen más no se implementan los métodos abstractos que más adelante en la clase ConjuntoArreglo<T>sí se implementaran.

## Explica las complicaciones de trabajar con arreglos y genéricos

Los arreglos de genérico, no se pueden inicializar directamente con *new T[n]*, se tiene que hacer de esta manera: *elementos = (T[ ]) new Object[n]* y eso genera la advertencia de "unchecked cast".

## Responde a las siguientes preguntas:

a. Al compilar el código salen unos warnings ¿A qué se debe?

   Es gracias a la manera de inicializar los arreglos genéricos como ya se había mencionado en la parte sobre las complicaciones de trabajar con arreglos y gnéricos.

b. ¿Consideras que está implementación de conjuntos es buena?

   Sí, es díficil trabajar con los arreglos, en especial para agregar elementos y en las operaciones de conjuntos unión e intersección, pero es una estructura de datos que funciona y emula el comportamiento de los conjutos desde la perspectiva matemática.

c. ¿Qué otros métodos puedes añadir en la clase Conjunto?

   Otras operaciones de conjuntos, como es el caso de la diferencia o diferencia relativa.

d. ¿Por qué existen métodos que reciben objetos de tipo Conjunto en lugar de objetos de tipo ConjuntoArreglo? ¿Qué ventajas tiene lo mencionado anteriormente?

   Permite que los que son de tipo Conjunto puedan trabajar con otro tipo de implementaciones distintas a las de ConjuntoArreglo.

e. ¿Para qué sirve la interfaz Iterable?

   Es para poder trabajar con los *for each*.

**Respuestas de la IA**

a. Al compilar el código salen unos warnings ¿A qué se debe?

Los warnings suelen aparecer por varias razones comunes en implementaciones de genéricos y arreglos:

*a*) Uso de genéricos con arrays: Java no pemite crear arrays de tipos genéricos directamente por temas de *type erasure*. Esto genera un warning.

*b*) Castings peligrosos: $x = (T[])newObject[10];$. Esto funciona pero el compilador te avisa de un *unchecked cast*.

*c*) Métodos que no usan genéricos correctamente o retornarn *Object* en lugar de $T$

En general, los warnings te indican que algo puede generar errores en tiempo de ejecución aunque compile.

b. ¿Consideras que está implementación de conjuntos es buena?

Depende de cómo esté hecha, pero algunas observaciones:

*a*) Ventajas:

- Es simple y fácil de entender.
- Funciona para conjuntos pequeños y demostraciones de conceptos.

*b*) Desventajas:

- Usar arrays limita el tamaño máximo; si quieres agregar más elementos, hay que redimensionar manualmente.
- Puede ser ineficiente para búsquedas, inseciones o borrados ($O(n)$).
- No previene duplicados de manera eficiente si se hace con un simple *for*.
- Los warnings de genéricos muestran que no es 100 % seguro tipo *type-safe*.

Conclusión: Está bien para aprendizaje, pero en producción conviene usar estructuras como *HashSet<T>* o *ArrayList<T>* con lógica de control de duplicados.

c. ¿Qué otros métodos puedes añadir en la clase Conjunto?

Algunos métodos útiles:

*a*) *boolean contiene(T elemento)* → Ver si un elemento está en el conjunto.

*b*) *Conjunto diferencia(Conjunto otro)* → Elementos de este conjunto que no están en el otro.

*c*) *boolean estaVacio()* → Ver si el conjunto tiene elementos.

*d*) *void clear()* → Vaciar el conjunto.

d. ¿Por qué existen métodos que reciben objetos de tipo Conjunto en lugar de objetos de tipo ConjuntoArreglo? ¿Qué ventajas tiene lo mencionado anteriormente?

Esto se llama programación contra interfaces o clases abstractas.

Ventaja: Tu método es más general y flexible. Por ejemplo: $publicConjuntounion(Conjuntootro)$.... Esto permite que otro pueda ser ConjuntoArreglo o cualquier otra implementación futura de Conjunto, sin tener que cambiar el código.

Si usas directamente ConjuntoArreglo, tu código queda acoplado y no podrás reutilizarlo con otras implementaciones.

Beneficio: Polimorfismo y mantenibilidad.

e. ¿Para qué sirve la interfaz Iterable?

*Iterable* permite que un objeto pueda ser recorrido con un *for-each*. Esto hace que tu clase sea compatible con las APIs de Java que esperan colecciones iterables, como Collections, streams,etc.

Básicamente, hace que tu conjunto sea más usable y consistente con las colecciones de Java.

## Análisis sobre las respuestas de la IA

Considero que fueron muy buenas respuestas las que me dio. En esta ocasión le especifiqué a ChatGPT que fuera menos formal, resumido y práctico y lo hizo a mi parecer satisfactoriamente, pues en el caso de los warnings me dio las principales razones por las cuales estos aparecen y lo que significan, sobre la implementación de conjuntos con arreglos fue muy concreto me dio claras ventajas y desventajas con alternativas mejores en otros contextos. En el caso de los métodos a implementar se quedo solo con la descripción de operaciones y aunque tuve que quitar algunos ejemplos como la unión, y la intersección que ya están implementados en la práctica los otros ejemplos siguen en el mismo sentido de conseguir más propiedades de los Conjuntos para tener un panorama mejor en la implementación. En las últimas preguntas fue igual de conciso, no se perdió en otros temas y respondio de conformidad a las preguntas.

## Explicación de los métodos de la clase *UtilArreglos*

1. *contieneNull*
    - Este método evalua todos los elementos del arreglo de entrada y si alguno de ellos es nulo regresa verdadero en caso contrario regresa falso.
2. *copiaArregloGenerico*
    - Este método podría utilizar primero crearArregloGenerico, para después solo asignar los mismos valores de los elemenos del arreglo de entrada al nuevo arreglo.
3. *crearArregloGenerico*
    - Este método evalua el tipo de elementos contenidos en el arreglo de entrada y después genera un nuevo arreglo vacío del mismo tipo que el anterior y de la longitud deseada.
4. *eliminarDuplicados*
    - Este método evalua todos los elementos del arreglo de entrada, y los compara tanto con los elementos previamente evaluados como con null, si resulta que son iguales, en un nuevo arreglo agregará los elementos que son distintos, en caso contrario regresa el mismo arreglo de entrada.
5. *tieneDuplicados*
    - Este método evalua y compara entre ellos a los elementos del arreglo de entrada, si alguno de ellos es igual a alguno de los elementos previamente evaluados, regresará verdadero, en caso contrario regresará falso.