

Reporte de la Práctica 2: Arreglos.

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

01 de septiembre del 2025

Este es el reporte de la segunda 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.

Me costo trabajo recordar como trabajar con arreglos. En el semestre pasado hice varias prácticas que involucraban listas, pero ninguna de arreglos, únicamente tenía presente la parte teórica y pensar en las implementaciones ya en Java me tomo un tiempo considerable.

Explica por qué no es posible usar el método `imprimeArreglo(Object[])` para imprimir los arreglos de tipo `int[]` y `double[]`. ¿Existe alguna forma de definir un método que imprima arreglos sin importar su tipo?

Se debe a que los objetos y los tipos primitivos en Java no se pueden convertir automáticamente a objetos y por esto es que se tienen que realizar métodos para cada uno de ellos por aparte.

Sí existe una manera de definir un método para imprimir arreglos sin importar su tipo. Utilizando `Arrays.toString`

Definición formal del algoritmo `reversaArreglo()`

Entrada: Un arreglo de tipo entero.

Salida: Un arreglo de tipo entero, con los mismos elementos que el arreglo de entrada, pero en orden invertido.

1. if `arreglo==null` then
2. return null
3. end if
4. Sea `reversa` un arreglo de tamaño `arreglo.length`
5. Para cada `i` con $0 \leq i < \text{arreglo.length}$ hacer:
6. `reversa[i] = arreglo [arreglo.length - 1 -i]`
7. return `reversa`

Número de operaciones: $T(n) = 9n + 6$. Su complejidad es $O(n)$

1. 1 operación
2. 1 operación
4. 2 operaciones
5. 1 operación + (3 operaciones por iteración)
6. 6 operaciones por iteración
7. 1 operación

Definición formal del algoritmo combinarArreglos()

Entrada: Un arreglo1 de tipo entero, y un arreglo2 de tipo entero.

Salida: Un arreglo generado de los dos originales que tiene cardinalidad igual a la suma de los arreglos parámetro y los mismos elementos que estos.

1. if arreglo1==null y arreglo2==null then
2. return null
3. end if
4. Sea combinacion un arreglo de tamaño arreglo1.length + arreglo2.length
5. Para cada i con $0 \leq i < \text{arreglo1.length}$ hacer:
6. combinacion[i] = arreglo1[i]
7. Para cada i con $0 \leq i < \text{arreglo2.length}$ hacer:
8. combinacion [arreglo1.length + i] = arreglo2[i]
9. return combinacion

Número de operaciones: $T(n) = 5n_1 + 7n_2 + 11$. Su complejidad es $O(n)$

1. 3 operaciones
2. 1 operación
4. 4 operaciones
5. 1 operación + (3 operaciones por iteración)
6. 2 operaciones por iteración
7. 1 operación + (3 operaciones por iteración)
8. 4 operaciones por iteración
9. 1 operación

Responde a las siguientes preguntas:

- a. ¿Es una mala práctica definir tres métodos distintos para imprimir arreglos? ¿A qué tipo de mala práctica correspondería esta decisión?
- Sí es una mala práctica, pues se repite código. Aunque para Java sí significa cada método una cosa distinta, se utiliza la misma lógica de programación para todos los métodos.
- b. ¿Por qué no es necesario incluir los archivos **Persona.java** y **Figura.java** para poder crear objetos de estas clases en el **main**?
- Java es un lenguaje compilado. Lo que significa que primero se compila el programa para después ejecutarlo. Los archivos .class son resultado de compilar el programa y es suficiente con tener estos archivos para ejecutar luego un programa que necesite de ellos. Es por esto que se puede no disponer de los archivos .java para crear objetos de esas clases en el main.

Respuestas de la IA

- a. ¿Es una mala práctica definir tres métodos distintos para imprimir arreglos? ¿A qué tipo de mala práctica correspondería esta decisión?
- Sí, es una mala práctica.
- Esta duplicación de código se conoce como “code duplication” o “violación del principio DRY (Don’t Repeat Yourself)”.
- Además, limita la escalabilidad: si mañana quieres imprimir un `char[]`, tendrías que escribir otro método más.
- La forma correcta es usar métodos genéricos (`<T>`), o bien sobrecargar con `Object[]` cuando se trate de clases. Para primitivos (`int`, `double`, etc.), Java provee `Arrays.toString(...)`.
- b. ¿Por qué no es necesario incluir los archivos **Persona.java** y **Figura.java** para poder crear objetos de estas clases en el **main**?
- Porque en Java:
- Cuando compilas el proyecto, cada archivo .java genera un .class.
 - El main no necesita ver el archivo fuente (.java) de Persona o Figura, solo necesita que la JVM tenga acceso a sus bytecodes (.class) en el classpath.
 - Por eso puedes instanciar objetos de esas clases aunque no esté el código fuente, mientras el compilador encuentre las clases compiladas.

En otras palabras:

- En tiempo de compilación, necesitas las definiciones de las clases (en .java o .class).
- En tiempo de ejecución, solo necesitas los .class.

¿Quieres que te arme una respuesta de examen corta y precisa (tipo párrafo breve con palabras clave), o una explicación larga más como la que te puse arriba?

Análisis sobre las respuestas de la IA

De nuevo ChatGPT fue la IA que utilicé y en esta ocasión su respuesta me fue de mayor utilidad, pues fue concisa y fueron respuestas apegadas a mis conocimientos e intuición sobre los temas que se me preguntaron en esta práctica.