UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA

TAREA

JAVA + MAVEN: ARRAYS

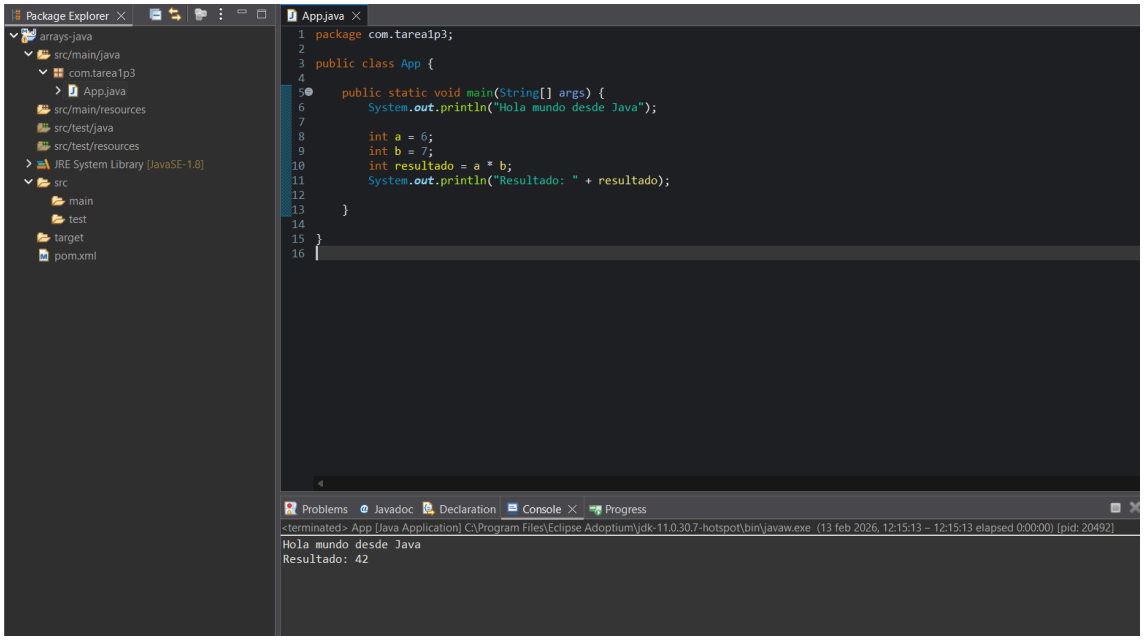
MILDRED HANANI PINEDA PINEDA
0905-22-5811

PROGRAMACION 3

FEBRERO 2026

```
Microsoft Windows [Versión 10.0.26200.6901]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\COMPUFIRE>java -version
openjdk version "11.0.30" 2026-01-20
OpenJDK Runtime Environment Temurin-11.0.30+7 (build 11.0.30+7)
OpenJDK 64-Bit Server VM Temurin-11.0.30+7 (build 11.0.30+7, mixed mode)

C:\Users\COMPUFIRE>
```

```
Microsoft Windows [Versión 10.0.26200.6901]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\COMPUFIRE>mvn -version
Apache Maven 3.9.12 (848fbb4bf2d427b72bdb2471c22fced7ebd9a7a1)
Maven home: C:\Maven\apache-maven-3.9.12
Java version: 11.0.30, vendor: Eclipse Adoptium, runtime: C:\Program Files\Eclipse Adoptium\jdk-11.0.30.7-hotspot
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"

C:\Users\COMPUFIRE>
```



4. Investigación: Arreglos En Java

4.1. ¿Cómo se declara un arreglo en Java?

✓ **Forma 1: Declaración y creación separada**

```
int[] numeros;
numeros = new int[5];
numeros[0] = 10;
numeros[1] = 20;
numeros[2] = 30;
```

✓ **Forma 2: Declaración e inicialización en una línea**

```
int[] valores = {1, 2, 3, 4, 5};
String[] nombres = {"Ana", "Juan", "María", "Pedro"};
double[] precios = {19.99, 25.50, 30.00};
```

✓ **Forma 3: Usando el operador new con valores**

```
int[] edades = new int[]{18, 25, 30, 45, 50};
String[] ciudades = new String[]{"Guatemala", "México", "Madrid"};
```

4.2. Métodos y utilidades principales para arreglos (java.util.Arrays)

4.2.1. Arrays.sort() - Ordenar arreglo

```
import java.util.Arrays;

int[] numeros = {5, 2, 8, 1, 9};
Arrays.sort(numeros);
System.out.println(Arrays.toString(numeros));
```

4.2.2. Arrays.binarySearch() - Buscar elemento

```
int[] numeros = {1, 3, 5, 7, 9, 11};
int indice = Arrays.binarySearch(numeros, 7);
System.out.println("Índice: " + indice);
```

4.2.3. Arrays.copyOf() - Copiar arreglo

```
java
int[] original = {1, 2, 3, 4, 5};
int[] copia = Arrays.copyOf(original, original.length);
System.out.println(Arrays.toString(copia));
```

4.2.4. Arrays.fill() - Llenar arreglo con un valor

```
int[] numeros = new int[5];
Arrays.fill(numeros, 7);
System.out.println(Arrays.toString(numeros));
```

4.2.5. Arrays.equals() - Comparar arreglos

```
int[] array1 = {1, 2, 3};
int[] array2 = {1, 2, 3};
boolean iguales = Arrays.equals(array1, array2);
System.out.println("Son iguales: " + iguales);
```

4.2.6. Arrays.toString() - Convertir arreglo a String

```
int[] numeros = {10, 20, 30, 40, 50};
String texto = Arrays.toString(numeros);
System.out.println(texto);
```

4.2.7. Arrays.stream() - Crear un stream desde el arreglo

```
java
int[] numeros = {1, 2, 3, 4, 5};
int suma = Arrays.stream(numeros).sum();
double promedio = Arrays.stream(numeros).average().getAsDouble();
System.out.println("Suma: " + suma + ", Promedio: " + promedio);
```

4.3. ¿Cómo se recorren los arreglos en Java?

✓ **Forma 1: For tradicional**

```
String[] frutas = {"Manzana", "Banana", "Naranja", "Uva"};

for (int i = 0; i < frutas.length; i++) {
  System.out.println("Posición " + i + ": " + frutas[i]);
}
```

- **Ventajas:** Acceso al índice, permite modificar elementos, se puede recorrer en orden inverso.

```
for (int i = frutas.length - 1; i >= 0; i--) {
    System.out.println(frutas[i]);
}
```

✓ **Forma 2: For-each**

```
String[] colores = {"Rojo", "Verde", "Azul", "Amarillo"};

for (String color : colores) {
    System.out.println(color);
}
```

- **Ventajas:** Sintaxis simple y legible, menos propenso a errores.
- **Desventajas:** No acceso al índice, no se puede modificar fácilmente.

✓ **Forma 3: Usando Streams**

```
import java.util.Arrays;

int[] numeros = {1, 2, 3, 4, 5};

Arrays.stream(numeros).forEach(num -> {
    System.out.println("Número: " + num);
});

Arrays.stream(numeros).forEach(System.out::println);

Arrays.stream(numeros)
    .filter(n -> n % 2 == 0)
    .map(n -> n * 2)
    .forEach(System.out::println);
```

- **Ventajas:** Permite operaciones funcionales, sintaxis moderna, fácil de encadenar operaciones.

4.4. **Diferencias entre Arreglos y ArrayList en Java**

✓ **Tamaño**

- **Arreglos:** Tamaño fijo que se define al crear el arreglo y no puede cambiar.
- **ArrayList:** Tamaño dinámico que crece automáticamente al agregar elementos.
- **Sintaxis de creación**

```
int[] arr = new int[5];
String[] nombres = {"Ana", "Juan", "María"};

ArrayList<Integer> list = new ArrayList<>();
ArrayList<String> nombresLista = new ArrayList<>();
```

✓ **Tipos de datos**

- **Arreglos:** Pueden almacenar tipos primitivos y objetos.
- **ArrayList:** Solo almacena objetos. Para primitivos usa clases envolventes.

✓ **Rendimiento**

- **Arreglos:** Más rápidos por acceso directo a memoria.
- **ArrayList:** Ligeramente más lentos por la gestión dinámica.

✓ **Métodos disponibles**

- **Arreglos:** Pocos métodos integrados, requiere java.util.Arrays para operaciones avanzadas.
- **ArrayList:** Muchos métodos útiles como add(), remove(), get(), set(), size(), contains(), etc.

✓ **Modificación de tamaño**

- **Arreglos:** No se puede cambiar el tamaño. Para agregar elementos se necesita crear un nuevo arreglo.
- **ArrayList:** Se pueden agregar y eliminar elementos fácilmente en cualquier momento.

✓ **Sintaxis de acceso y asignación**

```
int valor = arr[0];
arr[0] = 10;

int valor = list.get(0);
list.set(0, 10);
```

✓ **Ejemplo con Arreglo**

```
int[] numeros = new int[3];
numeros[0] = 10;
numeros[1] = 20;
numeros[2] = 30;

int longitud = numeros.length;

for (int num : numeros) {
    System.out.println(num);
}
```

- ✓ **Ejemplo con ArrayList**

```
import java.util.ArrayList;

ArrayList<Integer> numeros = new ArrayList<>();
numeros.add(10);
numeros.add(20);
numeros.add(30);
numeros.add(40);
numeros.add(50);

int tamaño = numeros.size();
int primero = numeros.get(0);
numeros.set(0, 15);
numeros.remove(2);

for (int num : numeros) {
    System.out.println(num);
}
```

- ✓ **Tipos primitivos vs Clases envolventes**

```
int[] numerosArreglo = {1, 2, 3, 4, 5};
double[] decimales = {1.5, 2.7, 3.9};

ArrayList<Integer> numerosLista = new ArrayList<>();
ArrayList<Double> decimalesLista = new ArrayList<>();

numerosLista.add(10);
int valor = numerosLista.get(0);
```

- ✓ **¿Cuándo usar cada uno?**
  - **Usa ARREGLOS cuando:**
    - Conoces el tamaño exacto y no cambiará
    - Necesitas máximo rendimiento
    - Trabajas con tipos primitivos
    - Necesitas arreglos multidimensionales
  - **Usa ARRAYLIST cuando:**
    - El tamaño puede cambiar durante la ejecución
    - Necesitas agregar o eliminar elementos frecuentemente
    - Quieres métodos convenientes
    - Necesitas flexibilidad sobre rendimiento
- ✓ **Ejemplo comparativo**

```
String[] estudiantesArray = new String[5];
estudiantesArray[0] = "Ana";
estudiantesArray[1] = "Juan";

ArrayList<String> estudiantesList = new ArrayList<>();
estudiantesList.add("Ana");
estudiantesList.add("Juan");
estudiantesList.add("Carlos");
estudiantesList.remove("Juan");
```

- ✓ **Resumen de Métodos Importantes**

```
import java.util.Arrays;
import java.util.ArrayList;

int[] arr = {5, 2, 8, 1, 9};
Arrays.sort(arr);
Arrays.toString(arr);
Arrays.binarySearch(arr, 8);
Arrays.fill(arr, 0);
Arrays.equals(arr1, arr2);
Arrays.copyOf(arr, 5);

ArrayList<String> list = new ArrayList<>();
list.add("elemento");
list.get(0);
list.set(0, "nuevo");
list.remove(0);
list.size();
list.contains("elemento");
list.clear();
list.isEmpty();
```