



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Rene Adrián Dávila Pérez

Asignatura: Programación orientada a objetos

Grupo: 01

No de Práctica(s): 08

Integrante(s): 321150819

*No. de lista o
brigada:*

Semestre: 2025-1

Fecha de entrega: Sábado 12 de octubre

Observaciones:

CALIFICACIÓN: _____

Índice

- 1) Introducción
- 2) Marco Teórico
- 3) Desarrollo
- 4) Resultados
- 5) Conclusiones
- 6) Referencias

1) Introducción

a) Ejercicio 1:

- Planteamiento del problema: Escribe un programa en Java para crear una interfaz llamada Ordenamiento que contenga el método ordenar. Luego, implementa esta interfaz en las clases BubbleSort y SelectionSort.
- Motivación: Hacer uso de interfaces dentro de java.
- Objetivos: Sobrescribir métodos y emplear interfaces para observar el comportamiento entre la clase y sus subclases.

b) Ejercicio 2:

- Planteamiento del problema: Escribe un programa en Java para crear una clase abstracta llamada Figura con el método calcularArea. Luego, crea las clases Círculo y Cilindro, que sobrescribirán el método calcularArea.
- Motivación: Emplear @Override para sobrescribir métodos.
- Objetivos: Sobrescribir métodos y observar la interacción entre la clase y sus subclases.

c) Práctica 81:

- Planteamiento del problema: Escribe un programa en Java para crear una interfaz llamada Ordenamiento, y las clases QuickSort y MergeSort, que implementen dicha interfaz.
- Motivación: Utilizar interfaces dentro de Java.
- Objetivos: Las clases deben de implementar la interfaz declarada.

d) Práctica 82:

- Planteamiento del problema: Escribe un programa en Java para crear la clase Empleado con los atributos nombre y rol, e implementar el método calcularSalario. Adicionalmente, crea las clases derivadas Gerente y Programador, con sus propios atributos, que sobrescriban el método calcularSalario.
- Motivación: Emplear @Override para sobrescribir métodos y crear una jerarquía de clases.
- Objetivos: Sobrescribir métodos y observar la interacción entre la clase y sus subclases.

2) Marco Teórico

- **String:** "Un objeto String representa una cadena de caracteres alfanuméricos con un valor inmutable, es decir, no se puede modificar después de su creación." [1]
- **Class:** "Las instancias de la clase Class representan clases e interfaces en una aplicación Java en ejecución." [2]
- **System.out:** "System.out es un objeto de la clase PrintStream, usado principalmente para mostrar texto en la consola." [3]
- **Override:** "Indica que una declaración de método está destinada a anular una declaración de método en un supertipo." [4]
- **Void:** "La clase Void es una clase de marcador de posición no instanciable para contener una referencia al objeto Class que representa la palabra clave Java Void." [5]
- **Interface:** "Una interfaz especifica el comportamiento de una clase proporcionando un tipo abstracto". [6]

3) Desarrollo

a) Ejercicio 1:

El código comienza definiendo el paquete `mx.unam.fi.poo.g1.p8`. Se define la interfaz `Ordenamiento` que tiene un método abstracto `ordenar`, el cual recibe un arreglo de enteros como parámetro.

Se declara la clase `BubbleSort`, que implementa la interfaz `Ordenamiento`. El método `ordenar` utiliza el algoritmo de burbuja para ordenar el arreglo. Este algoritmo itera sobre el arreglo y compara elementos adyacentes,

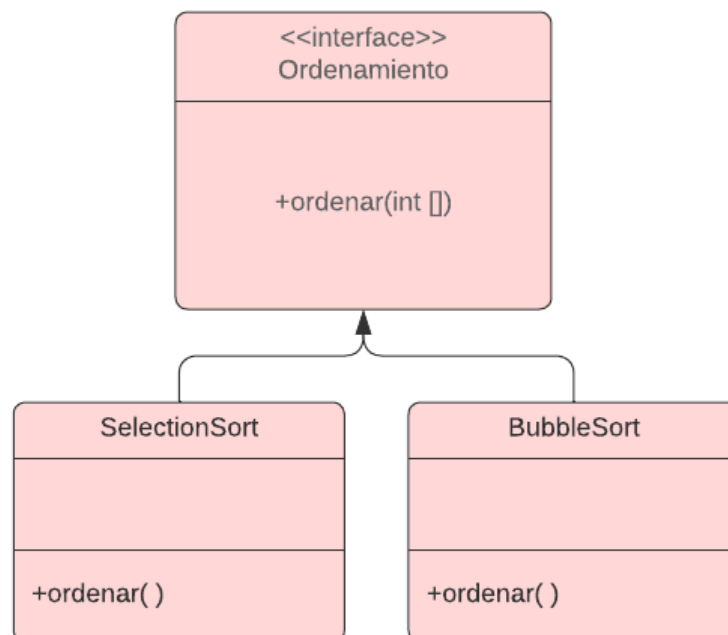
intercambiándolos si están en el orden incorrecto, repitiendo el proceso hasta que todos los elementos estén ordenados.

Se declara la clase SelectionSort, que también implementa la interfaz Ordenamiento. El método ordenar utiliza el algoritmo de selección, que selecciona el elemento más pequeño de la parte no ordenada del arreglo y lo intercambia con el primer elemento no ordenado, repitiendo este proceso hasta que todo el arreglo está ordenado.

Dentro de la clase principal Ejercicio0, se define el método main, donde se crea un arreglo de enteros con los valores {4, 2, 0, 3, 1, 6, 8}. Primero se crea un objeto BubbleSort, se llama al método ordenar con el arreglo y luego se imprime el arreglo ordenado usando el método imprime. Después se crea un objeto SelectionSort, se vuelve a ordenar el arreglo y se imprime nuevamente.

El método estático imprime recibe un arreglo de enteros y lo imprime elemento por elemento, separados por un espacio, seguido de una línea en blanco.

A continuación se muestran los diagramas de las clases.



Pruebas:

0	0
1	1
2	2
3	3
4	4
6	6
8	8

b) Ejercicio 2:

El código comienza definiendo el paquete `mx.unam.fi.poo.g1.p8`. Se declara la clase abstracta `Figura` que contiene dos métodos abstractos: `dibujar`, que no recibe parámetros, y `calculaArea`, que regresa un valor de tipo `double`. Ambas funciones deben ser implementadas por las clases que hereden de esta clase.

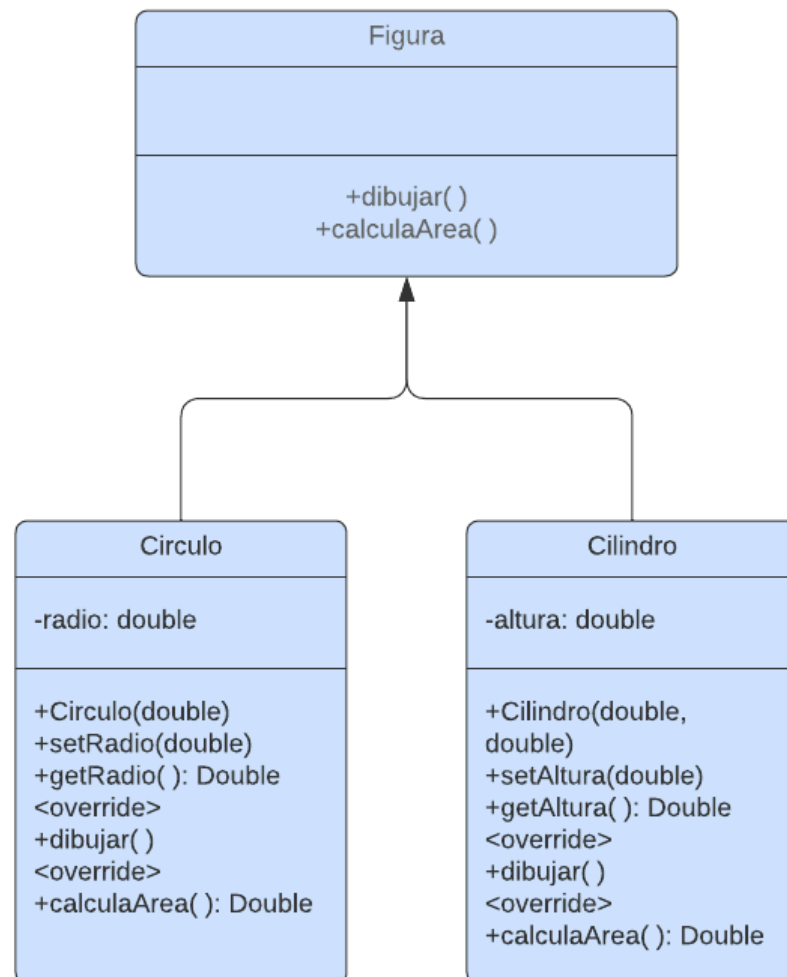
Se define la clase `Circulo`, que extiende de la clase `Figura`. Esta clase tiene un atributo privado `radio` de tipo `double`. El constructor de `Circulo` recibe el valor del radio y lo asigna mediante el método `setRadio`. Además, la clase contiene los métodos `setRadio` y `getRadio` para establecer y obtener el valor del radio, respectivamente. Se sobrescriben los métodos `dibujar`, que imprime un mensaje indicando que en el futuro se usarán gráficos en Java, y `calculaArea`, que regresa el área del círculo utilizando la fórmula $\pi * \text{radio}^2$.

Se define la clase `Cilindro`, que extiende de `Circulo`. Esta clase tiene un atributo adicional `altura` de tipo `double`. Su constructor recibe los valores de radio y altura, llamando al constructor de la clase padre para inicializar el radio y utilizando el método `setAltura` para asignar la altura. La clase también tiene los métodos `setAltura` y `getAltura` para establecer y obtener el valor de la altura. Se sobrescriben los métodos `dibujar`, que imprime un mensaje indicando que se trata de un cilindro abstracto, y `calculaArea`, que calcula el área del cilindro sumando dos veces el área del círculo base y el área lateral, dada por la fórmula $2 * \pi * \text{radio} * \text{altura}$.

Dentro de la clase principal Ejercicio1, se define el método main, que crea un objeto de tipo Circulo con un radio de 7.0 y un objeto de tipo Cilindro con un radio de 4.0 y una altura de 9.0. Ambos objetos se pasan al método dibujaCalcula, que llama a los métodos dibujar y calculaArea para cada objeto, mostrando el área calculada en la consola.

El método estático dibujaCalcula recibe un objeto de tipo Figura, llama al método dibujar de dicho objeto, luego calcula su área con el método calculaArea e imprime el resultado.

A continuación se muestran los diagramas de las clases.



Pruebas:

Más adelante van a usar gráficos en Java
Area: 153.93804002589985

Un Cilindro abstracto...
Area: 326.7256359733385

c) Práctica 81:

c.1) Práctica 81:

El código comienza definiendo el paquete `mx.unam.fi.poo.g1.p8`. Se importa el paquete completo.

Se declara la clase principal `Practica81`, la cual contiene el método `main`. Este método tiene como objetivo ejecutar todo el funcionamiento de la aplicación.

Dentro del método `main`, se crea un arreglo de enteros con los valores {50, 32, 27, 9, 13, 19, 20}. Se crea un objeto de tipo `Quicksort` que implementa la interfaz `Ordenamiento`. Primero, se imprime el arreglo original utilizando el método `imprime`. Luego, el arreglo es ordenado con el método `ordenar` del objeto `Quicksort`, y se imprime el arreglo ordenado.

Después, se crea un segundo arreglo con los mismos valores iniciales. Se crea un objeto de tipo `MergeSort`, también de la interfaz `Ordenamiento`. El segundo arreglo se ordena con el método `ordenar` del objeto `MergeSort`, y finalmente se imprime el arreglo ordenado.

El método `imprime` es un método estático que recibe un arreglo de enteros y lo imprime elemento por elemento, seguidos de un espacio, y finaliza con una línea en blanco.

c.2) Ordenamiento:

El código define la interfaz `Ordenamiento` dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta interfaz contiene un único método llamado `ordenar`, que debe ser implementado por cualquier clase que la implemente.

El método ordenar recibe como parámetro un arreglo de enteros, el cual será ordenado de acuerdo al algoritmo de ordenamiento que se implemente en las clases que utilicen esta interfaz.

Esta interfaz sirve como un contrato para asegurar que cualquier clase que la implemente defina un comportamiento específico para ordenar un arreglo de enteros.

c.3) QuickSort:

El código define la clase Quicksort dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase implementa la interfaz Ordenamiento, lo que obliga a la implementación del método ordenar, que recibe un arreglo de enteros como parámetro.

El método ordenar invoca al método quickSort, que implementa el algoritmo de ordenamiento rápido (Quicksort). Este algoritmo funciona de manera recursiva, dividiendo el arreglo en subarreglos más pequeños utilizando un pivote. Luego, los elementos menores que el pivote se colocan a su izquierda, y los mayores, a su derecha.

El método quickSort recibe como parámetros el arreglo a ordenar, el índice inicial y el índice final del subarreglo que se está procesando. Si el índice inicial es menor que el final, se llama al método particion para encontrar la posición del pivote y se aplica recursivamente el algoritmo a las dos mitades resultantes.

El método particion selecciona el último elemento del subarreglo como pivote. Recorre los elementos del subarreglo comparándolos con el pivote y, cuando encuentra uno que es menor o igual, lo intercambia con el elemento en la posición adecuada. Finalmente, coloca el pivote en su posición correcta en el arreglo y devuelve su índice.

c.4) MergeSort:

El código define la clase MergeSort dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase implementa la interfaz Ordenamiento, lo

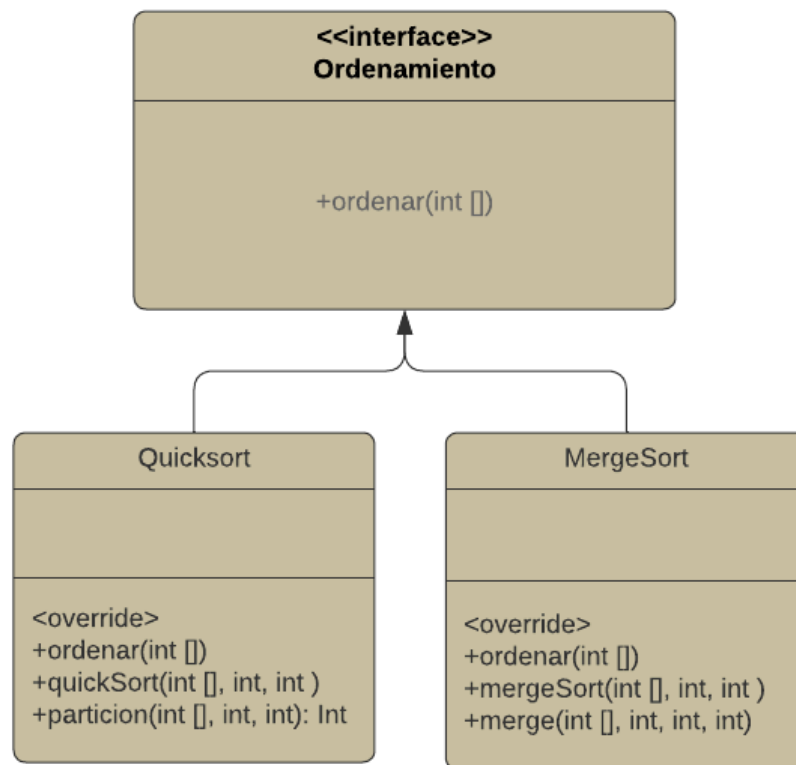
que obliga a la implementación del método ordenar, el cual recibe un arreglo de enteros como parámetro.

El método ordenar invoca al método mergeSort, que aplica el algoritmo de ordenamiento por mezcla (MergeSort). Este algoritmo es recursivo y divide el arreglo en subarreglos más pequeños, los cuales son ordenados individualmente y luego fusionados.

El método mergeSort recibe como parámetros el arreglo a ordenar, el índice inicial y el índice final del subarreglo que se está procesando. Si el índice inicial es menor que el índice final, se calcula el índice medio para dividir el arreglo en dos mitades. Luego, se llama recursivamente a mergeSort para ordenar ambas mitades, y finalmente, se fusionan las sublistas ordenadas utilizando el método merge.

El método merge toma las dos mitades ordenadas y las combina en una sola lista ordenada. Para lograrlo, crea dos arreglos temporales, izquierda y derecha, que contienen los elementos de cada mitad. Luego, recorre ambos arreglos comparando los elementos y los coloca en el arreglo original en el orden correcto. Si quedan elementos en alguno de los arreglos temporales después de la comparación, se copian directamente al arreglo original.

A continuación se muestran los diagramas de las clases.



Pruebas:

Arreglo original:

50
32
27
9
13
19
20

Por QuickSort:

9
13
19
20
27
32
50

Por MergeSort:

9
13
19
20
27
32
50

d) Práctica 82:

d.1) Práctica 82:

El código define la clase principal Practica82 dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase contiene el método `main`, que es el punto de entrada para ejecutar todo el funcionamiento de la aplicación.

En el método `main`, primero se muestra un mensaje de bienvenida en la consola seguido de los datos del equipo. Luego, se crea un objeto de la clase `Gerente` con el nombre "Haziél Alvarez" y el rol "Gerente". A continuación, se imprimen los datos del gerente: su nombre y rol.

Después, se crea un objeto de la clase `Programador` con el nombre "Elena Alvarez" y el rol "Programador", y de igual manera, se imprimen sus datos: nombre y rol.

Finalmente, se muestra el salario de ambos empleados calculado en función de las horas trabajadas. El salario del gerente se calcula para 40 horas, y el salario del programador para 50 horas, utilizando el método `calcularSalario` de cada objeto.

d.2) Empleado:

El código define la clase `Empleado` dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase contiene dos atributos privados: `nombre` y `rol`, ambos de tipo `String`.

La clase tiene un constructor que recibe como parámetros el nombre y el rol del empleado, y los inicializa utilizando los métodos `setNombre` y `setRol`.

Se definen los métodos `getNombre` y `getRol`, que devuelven los valores de los atributos `nombre` y `rol`, respectivamente. También se incluyen los métodos `setNombre` y `setRol`, que permiten establecer o modificar los valores de estos atributos.

El método `calcularSalario` recibe como parámetro la cantidad de horas trabajadas y regresa el salario del empleado. El cálculo se basa en una tarifa fija de 500 unidades monetarias por hora trabajada.

d.3) Gerente:

El código define la clase `Gerente` dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase extiende la clase `Empleado`, lo que significa que hereda todos los atributos y métodos de `Empleado`.

El constructor de `Gerente` recibe como parámetros el nombre y el rol del gerente, y llama al constructor de la clase padre (`Empleado`) utilizando la palabra clave `'super'` para inicializar estos atributos.

Se sobrescribe el método `calcularSalario` para ajustar el cálculo del salario de un gerente. A diferencia de la clase `Empleado`, en la que el salario se calcula multiplicando las horas trabajadas por 500, en la clase `Gerente` el salario se calcula multiplicando las horas trabajadas por 300.

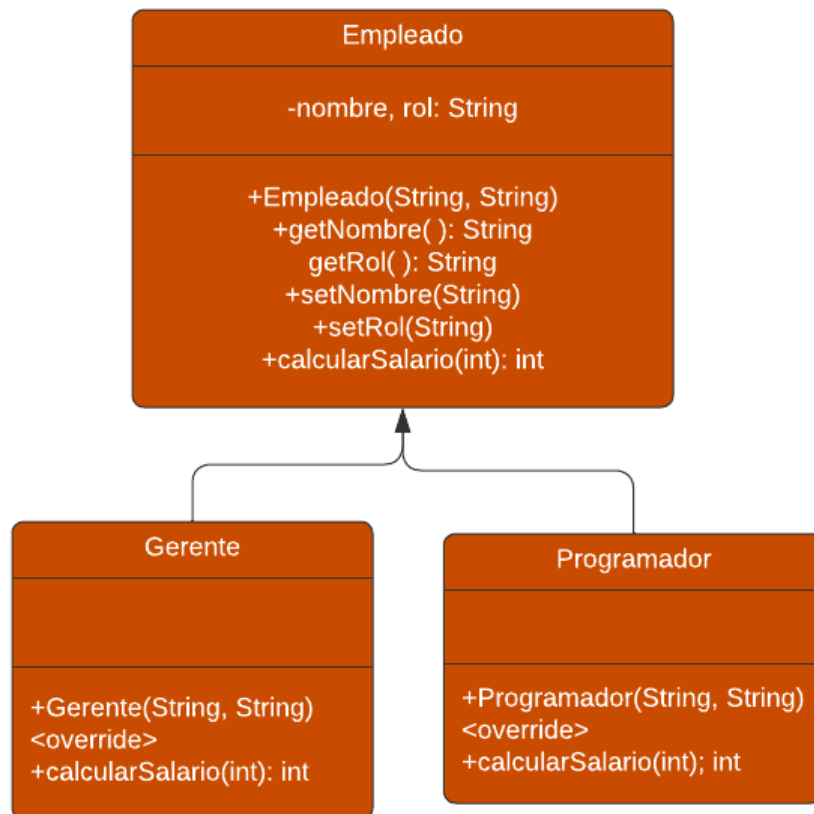
d.4) Programador:

El código define la clase `Programador` dentro del paquete `mx.unam.fi.poo.g1.p8`. Esta clase extiende la clase `Empleado`, lo que le permite heredar sus atributos y métodos.

El constructor de `Programador` recibe como parámetros el nombre y el rol del programador, y utiliza la palabra clave `'super'` para llamar al constructor de la clase padre (`Empleado`) e inicializar estos atributos.

Se sobrescribe el método calcularSalario para ajustar el cálculo del salario de un programador. A diferencia de la clase Empleado, donde el salario se calcula multiplicando las horas trabajadas por 500, en la clase Programador el salario se calcula multiplicando las horas trabajadas por 600.

A continuación se muestran los diagramas de las clases.



Pruebas

-- Bienvenid@ --

Datos de nuestro equipo:

Datos del Gerente:

Nombre: Haziel Alvarez

Rol: Gerente

Datos del Programador:

Nombre: Elena Alvarez

Rol: Programador

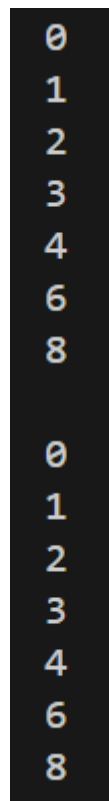
Su salario es:

Gerente: \$12000

Programador: \$30000

Resultados

a) Ejemplo 0:



0
1
2
3
4
6
8

0
1
2
3
4
6
8

Descripción: Este es el resultado por BubbleSort y SelectionSort para un arreglo de [4, 2, 0, 3, 1, 6, 8].

b) Ejemplo 1:

```
Más adelante van a usar gráficos en Java
Area: 153.93804002589985
Un Cilindro abstracto...
Area: 326.7256359733385
```

Descripción: Imprime el área de un círculo con radio 7 y el área de un cilindro con radio 4 y altura 9.

c) Práctica 8_1:

```
Arreglo original:
50
32
27
9
13
19
20

Por QuickSort:
9
13
19
20
27
32
50

Por MergeSort:
9
13
19
20
27
32
50
```


Descripción: Este es el resultado por BubbleSort y MergeSort para un arreglo de [50, 32, 27, 9, 13, 19, 20]. Sin olvidar que muestra el arreglo original sin cambios.

d) Práctica 8_2:

```
-- Bienvenid@ --  
Datos de nuestro equipo:  
  
Datos del Gerente:  
Nombre: Haziel Alvarez  
Rol: Gerente  
  
Datos del Programador:  
Nombre: Elena Alvarez  
Rol: Programador  
  
Su salario es:  
Gerente: $12000  
Programador: $30000
```

Descripción: Se muestran los datos del gerente y programador, nombre y rol. Por último, se muestran sus salarios.

4) Conclusiones

La implementación de las interfaces y clases abstractas dentro de java permiten estructurar programas de manera eficiente, facilitando su reutilización. Se destacó la aplicación del concepto de polimorfismo, la jerarquía de clases y la flexibilidad que las interfaces proporcionan.

5) Referencias

[1] "Tutorial de Java - La clase String". Instituto Tecnológico de la Paz. Accedido el 14 de agosto de 2024. [En línea]. Disponible: <http://www.itlp.edu.mx/web/java/Tutorial%20de%20Java/Cap3/string.html#:~:text=Ja>

va%20posee%20gran%20capacidad%20para,cadena%20cuyo%20tamaño%20puede
%20variar

[2] “Class” (Java Platform SE 8). Moved. Accedido el 12 de septiembre de 2024. [En línea]. Disponible: <https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html>

[3] “System.out”. W3Api. Accedido el 14 de agosto de 2024. [En línea]. Disponible: https://www.w3api.com/Java/System/out/#google_vignette

[4] “Annotation Type Override” (Java Platform SE 8). Moved. Accedido el 03 de octubre de 2024. [En línea]. Disponible: <https://docs.oracle.com/javase/8/docs/api/java/lang/Override.html>

[5] “Void” (Java Platform SE 8). Moved. Accedido el 17 de septiembre de 2024. [En línea]. Disponible: <https://docs.oracle.com/javase/8/docs/api/java/lang/Void.html>

[6] “What is Interface in Java?”. Simplilearn. Accedido el 10 de octubre de 2024. [En línea]. Disponible: <https://www.simplilearn.com/tutorials/java-tutorial/java-interface#:~:text=In%20Java%20C%20an%20interface%20specifies,in%20Java%20to%20achieve%20abstraction.>