

Sobrecargas

Contenido

Sobrecargas (overloading)

- ❖ ¿Qué es una sobrecarga?
- ❖ Reglas para la sobrecarga.
- ❖ Sobrecarga de métodos.
- ❖ Sobrecarga de constructor.

Manejo de String

- ❖ Creación y comparación.
- ❖ Métodos.
- ❖ La clase StringBuilder.

Enumerados

- ❖ Concepto.
- ❖ Definición y uso.



¿Qué es una sobrecarga?

En Java, la sobrecarga (**overloading**) se refiere a la capacidad de definir múltiples miembros con el mismo nombre, pero con diferentes listas de parámetros.

Las sobrecargas permiten que una clase ofrezca variantes del mismo comportamiento que se adapten a diferentes **tipos** o **cantidades** u **orden** de los argumentos.

En Java se pueden sobrecargar los **métodos** y el **constructor**.

Reglas para la sobrecarga

Lista de parámetros diferente:

Para que los métodos se consideren sobrecargados, deben tener una lista de parámetros diferente (**número** y/o **tipo** y/u **orden**).

Tipo de retorno: El tipo de retorno puede ser diferente, pero no es suficiente por sí solo para diferenciar los métodos sobrecargados.

Modificadores: Pueden tener diferentes modificadores de acceso (public, private, protected), pueden ser estáticos o de instancia y pueden lanzar diferentes excepciones, pero esto no es suficiente para diferenciarlos.

Sobrecarga de métodos

CANTIDAD DE PARÁMETROS

```
public void sobrecarga()  
{  
    // CÓDIGO  
}  
public int sobrecarga(int a)  
{  
    return a;  
}
```

TIPOS DE PARÁMETROS

```
public int sobrecarga(int a)  
{  
    return a;  
}  
public void sobrecarga(float a)  
{  
    // CÓDIGO  
}
```

ORDEN DE PARÁMETROS

```
public void sobrecarga(int a, float b)  
{  
    // CÓDIGO  
}  
public void sobrecarga(float a, int c)  
{  
    // CÓDIGO  
}
```

Sobrecarga de constructor



```
public class Persona
{
    private String apellido;
    private String nombre;
    private int edad;

    public Persona()
    {

    }

    public Persona(String apellido)
    {

    }

    public Persona(String apellido, String nombre)
    {

    }

    public Persona(String apellido, String nombre, int edad)
    {

    }
}
```

VALORES INICIALES

```
public Persona()
{
    this.apellido = "Sin apellido";
    this.nombre = "Sin nombre";
    this.edad = 0;
}

public Persona(String apellido)
{
    this.apellido = apellido;
    this.nombre = "Sin nombre";
    this.edad = 0;
}

public Persona(String apellido, String nombre)
{
    this.apellido = apellido;
    this.nombre = nombre;
    this.edad = 0;
}

public Persona(String apellido, String nombre, int edad)
{
    this.apellido = apellido;
    this.nombre = nombre;
    this.edad = edad;
}
```

REUTILIZANDO CÓDIGO





```
public Persona()
{
    this.apellido = "Sin apellido";
    this.nombre = "Sin nombre";
    this.edad = 0;
}

public Persona(String apellido)
{
    this();
    this.apellido = apellido;
}

public Persona(String apellido, String nombre)
{
    this(apellido);
    this.nombre = nombre;
}

public Persona(String apellido, String nombre, int edad)
{
    this(apellido, nombre);
    this.edad = edad;
}
```

Buenas prácticas

	Utilizar nombres de parámetros descriptivos.
	Evitar variar arbitrariamente los nombres de los parámetros en las sobrecargas. Si un parámetro en una sobrecarga representa la misma entrada de un parámetro en otra sobrecarga, los parámetros deben tener el mismo nombre.
	Evitar modificar el orden de los parámetros en los métodos sobrecargados. Los parámetros con el mismo nombre deben aparecer en la misma posición en todas las sobrecargas.
	No tener sobrecargas con parámetros en la misma posición y tipos similares pero con semántica diferente.

Ejercitación

Manejo de String

Manejo de String - Creación

Literal de Cadena

Cuando se crea una cadena usando **literales** `String s1 = "algo";` Java utiliza un mecanismo llamado **String Pool** o *piscina de cadenas*.

- ❖ **String Pool:** Es una sección especial de la memoria en el *Heap* donde las cadenas de texto literales se almacenan. Si se crea una cadena literal y otra cadena con el mismo valor ya existe en el pool, se reutiliza la referencia a la cadena existente en lugar de crear una nueva.
- ❖ **Eficiencia:** Es más eficiente en términos de memoria porque evita la creación de múltiples instancias de cadenas con el mismo contenido.

Manejo de String - Creación

Constructor de Cadena

Cuando se crea una cadena usando el constructor, `String s2 = new String(original:"algo");` se crea un nuevo objeto String en la memoria del *Heap*, independientemente de si una cadena con el mismo valor ya existe en el *String Pool*.

- ❖ **Memoria:** Siempre crea un nuevo objeto String en el Heap.
- ❖ **Uso:** Generalmente no es necesario a menos que haya una razón específica para necesitar una nueva instancia de la cadena.

Manejo de String - Comparación

Comparación de referencias y contenidos

- ❖ **Comparación de referencias (==):** Compara si dos **referencias** de objetos apuntan al mismo objeto en memoria.
- ❖ **Comparación de contenido (equals()):** Compara el **contenido** de las cadenas.

```
String s1 = "algo";  
String s2 = new String(original:"algo");  
  
boolean mismaReferencia = (s1 == s2); // false, diferentes referencias  
boolean mismoContenido = s1.equals(s2); // true, mismo contenido
```



Manejo de String - Métodos

```
// Concatenación
String str1 = "Hola";
String str2 = "Mundo";
String str3 = str1 + " " + str2;
// "Hola Mundo"
```

```
// Longitud
int length = str3.length();
// 10
```

```
// Subcadena
String substr = str3.substring(beginIndex:0, endIndex:4);
// Hola
```

```
// Reemplazo
String replacedStr = str3.replace(target:"Hola", replacement:"Adiós");
// "Adiós mundo"
```

```
// Comparación
boolean isEqual = str1.equals(anObject:"hola"); // false
boolean isEqualIgnoreCase = str1.equalsIgnoreCase(anotherString:"HOLA"); // true
int comparison = str1.compareTo(str2); // valor negativo, positivo o 0
```

```
// Transformación
String upper = str3.toUpperCase();
// "HOLA MUNDO"
String lower = str3.toLowerCase();
// "hola mundo"
```

```
// Búsqueda
int index = str3.indexOf(str:"Mundo"); // 5
int lastIndex = str3.lastIndexOf(str:"o"); // 7
boolean contains = str3.contains(s:"Hola"); // true
```

```
// División
String[] parts = str3.split(regex:" ");
// ["Hola", "Mundo"]
```

```
// Formateo
String formatted = String.format(format:"Hola, %s!", ...args:"Mundo");
// Hola, Mundo
```

```
// Eliminación de espacios en blanco
String trimmed = " sin espacios ".trim();
// "sin espacios"
```

La clase StringBuilder

StringBuilder es una clase en Java que se utiliza para crear y manipular cadenas de texto de manera eficiente, especialmente cuando se requiere realizar muchas modificaciones a la cadena.

A diferencia de la clase *String*, que es inmutable, ***StringBuilder*** permite modificar el contenido sin crear nuevos objetos, lo cual mejora el rendimiento.

Su uso puede mejorar significativamente el rendimiento en comparación con el uso de la clase *String* inmutable, especialmente en bucles y operaciones repetitivas de concatenación o modificación de cadenas.

La clase StringBuilder

Características de StringBuilder

- ❖ **Mutable:** Permite cambiar su contenido sin crear nuevos objetos.
- ❖ **Eficiente:** Es más eficiente que String cuando se realizan muchas concatenaciones o modificaciones en la cadena.
- ❖ **No Seguro para Hilos:** No es sincronizado, lo que significa que no es seguro para su uso en entornos multihilo. Para entornos multihilo, se puede usar ***StringBuffer***, que es sincronizado.



La clase StringBuilder - Métodos

```
// Crear un StringBuilder
StringBuilder sb = new StringBuilder(str:"Hola");
```

```
// Substring
String sub = sb.substring(start:0, end:4);
// "Hola"
```

```
// Agregar
sb.append(str:" Mundo");
sb.toString(); // "Hola Mundo"
```

```
// Longitud
int length = sb.length();
// 10
```

```
// Eliminar
sb.delete(start:5, end:11);
sb.toString(); // "Hola Mundo"
```

```
// Insertar
sb.insert(offset:5, str:"Querido ");
sb.toString(); // "Hola Querido Mundo"
```

```
// Reemplazar
sb.replace(start:5, end:12, str:"Amigo");
sb.toString(); // "Hola Amigo Mundo"
```

```
// Revertir
sb.reverse();
sb.toString(); // "odnuM aloH"
```


Enumerados

Enumerados

Los enumerados (enums) en Java son una característica del lenguaje que permite definir un conjunto de constantes con nombre.

Un enumerado se define utilizando la palabra clave **enum**.

Son útiles para representar un conjunto fijo de valores conocidos, como los días de la semana, los colores, los estados de un proceso, etc.

Definición y uso de un Enumerado

```
enum Velocidad
{
    MINIMA,
    MEDIA,
    MAXIMA
}
```

```
Velocidad velocidad = Velocidad.MAXIMA;
System.out.println(velocidad); // MAXIMA
```

```
for(Velocidad v : Velocidad.values())
{
    sb.append(v);
    sb.append(str: " ");
}

System.out.println(sb.toString());
// MINIMA MEDIA MAXIMA
```

Enumerados

Los enumerados pueden tener campos, métodos y constructor, igual que las clases.

```
enum Velocidad
{
    MINIMA(velocidad:20),
    MEDIA(velocidad:80),
    MAXIMA(velocidad:120);

    private final int velocidad;

    private Velocidad(int velocidad)
    {
        this.velocidad = velocidad;
    }

    public int getVelocidad()
    {
        return this.velocidad;
    }
}
```

```
for(Velocidad v : Velocidad.values())
{
    sb.append(str:"La velocidad ");
    sb.append(v);
    sb.append(str:" es de ");
    sb.append(v.getVelocidad());
    sb.append(str:" km/h.\n");
}

System.out.println(sb.toString());
// La velocidad MINIMA es de 20 km/h.
// La velocidad MEDIA es de 80 km/h.
// La velocidad MAXIMA es de 120 km/h.
```

Ejercitación