

# Java Standard Web Programming

Módulo 5

# Genéricos

# Introducción

Java es un lenguaje de tipado fuerte. Se debe **asignar un tipo de dato a cada atributo** y eso permite que, en tiempo de compilación, no se asigne un valor distinto al tipo de dato declarado.

Esto tiene sus ventajas, pero ¿qué pasa si en algún momento necesitamos una **estructura (clase) genérica** que nos ayude a **solventar varios problemas**, como una clase con dos atributos?

**Por ejemplo:** tenemos una clase que contiene dos atributos de tipo entero (código y número). Este tipo de objeto puede ser muy útil no solo para teléfono sino para otros casos, Documento, Domicilio, etc. **Tendríamos que crear N cantidad de clases para cada caso de uso.**

```
public class Telefono {  
    private int codigo;  
    private int numero;  
  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public void setCodigo(int codigo) {  
        this.codigo = codigo;  
    }  
  
    public int getNumero() {  
        return numero;  
    }  
  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
}
```

¿Qué ocurre si necesitamos un código que sea un **string** o un número que sea un **double**?

Sería posible resolver declarando los tipos de datos como **Object** ya que, por polimorfismo, se convierte en cualquiera de sus hijos, incluyendo estructuras nuevas y todos los datos primitivos, lo que exigiría más código y esfuerzo de nuestro lado para castear cada caso de uso o preguntar si el objeto es una instancia del tipo de dato deseado.

El problema que tenemos con **Object** es que debemos hacer unas validaciones para que no asignen el dato incorrecto.

```
public class DosAtributos {  
    private Object codigo;  
    private Object numero;  
  
    public Object getCodigo() {  
        return codigo;  
    }  
  
    public void setCodigo(Object codigo) {  
        this.codigo = codigo;  
    }  
  
    public Object getNumero() {  
        return numero;  
    }  
  
    public void setNumero(Object numero) {  
        this.numero = numero;  
    }  
}
```

## Ejemplo

```
DosAtributos telefono = new DosAtributos();
telefono.setCodigo(50);
telefono.setNumero(124478);

telefono.setCodigo("codigo");
telefono.setNumero("");

if (!(telefono.getCodigo() instanceof Integer)) {
    System.out.println("Error: el codigo debe ser numericos");
}

if (!(telefono.getNumero() instanceof Integer)) {
    System.out.println("Error: el numero debe ser numericos");
}

try {
    int codigo = Integer.parseInt((String) telefono.getCodigo())
    int numero = Integer.parseInt((String) telefono.getNumero())
} catch (Exception e) {
    System.out.println("Error: los datos deben ser numericos");
}
```

Aquí es donde entran las **clases genéricas** - o los también llamados **tipos parametrizados** -, incorporadas desde la *versión 5*. Permiten crear clases, interfaces y métodos en los que **los tipos de datos sobre los que queremos operar se envían como argumentos al instanciar la clase**. De esta forma la clase trabajará el dato que le fue enviado.

---


```
Public class GenericaDosAtributos<K, V> {  
  
    private K codigo;  
    private V numero;  
  
    public K getCodigo() {  
        | return codigo;  
    }  
  
    public void setCodigo(K codigo) {  
        | this.codigo = codigo;  
    }  
  
    public V getNumero() {  
        | return numero;  
    }  
  
    public void setNumero(V numero) {  
        | this.numero = numero;  
    }  
}
```

# Nomenclatura

Letra	Valor	Descripción
E	Element	Usado bastante por Java Collections Framework.
K	Key	Clave, usado en mapas.
N	Number	Para números.
T	Type	Representa un tipo, es decir, un objeto.
V	Value	Representa el valor, también se usa en mapas.
S, U, V, etc.		Usado para representar otros tipos.




# Instancias



```
GenericosAtributos<Integer, Integer> telefono = new GenericosAtributos<Integer, Integer>();  
GenericosAtributos<String , Integer> documento = new GenericosAtributos<String, Integer>();  
GenericosAtributos<String, String> direccion = new GenericosAtributos<String, String>();
```

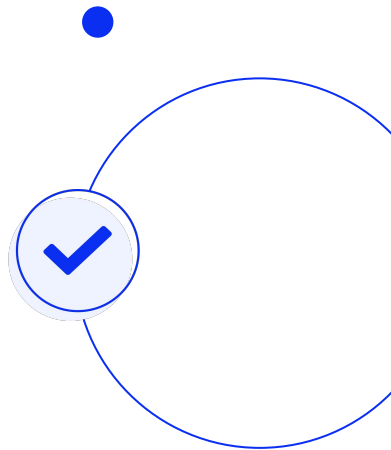
```
GenericosAtributos<Integer, Integer> telefono = new GenericosAtributos<>();  
GenericosAtributos<String , Integer> documento = new GenericosAtributos<>();  
GenericosAtributos<String, String> direccion = new GenericosAtributos<>();
```





## Ventajas

- Lo que hace a las clases genéricas una gran utilidad es que **comprueban los tipos de datos en tiempo de compilación** según el parámetro enviado a la clase al momento de declararlo.
- También **ayudan a reducir el código** eliminando, las comprobaciones de tipos y los casting excesivos.
- Y parte de lo más importante es la **reutilización de código**.



## Ejemplo

```
GenericaDosAtributos<Integer, Integer> telefono = new GenericaDosAtributos<>();  
telefono.setCodigo("CABA");  
telefono.setNumero(50124478);
```



**¡Sigamos  
trabajando!**