



# Clases Wrapper y Generics

## Clases Wrapper

Son clases envoltorio, que representan a un dato primitivo en forma de Objetos

- Anidan un valor primitivo para darle funcionalidad.
- Proveen de métodos de conversión a tipos compatibles
- Métodos de validación
- Equivalen a un tipo de referencia o clase.
- Cada primitivo posee clase su equivalente

### Tipos de referencia a cada primitivo

Tipo de dato primitivo	Clase Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

### Algunos Métodos Clase Integer

Método estáticos	Tipo devuelto
Integer.parseInt(String s)	int
Integer.valueOf(String s)	Integer
Integer.valueOf(int i)	Integer
Integer.toString(int i)	String

### Algunos Métodos Clase Integer

Método Concretos	Tipo devuelto
compareTo(Integer value)	int
equal(Integer value)	boolean
intValue()	int
longValue()	long
shortValue()	short
doubleValue()	double
floatValue()	float

## Generics

- Además del polimorfismo, una característica que permite a la plataforma Java tratar homogéneamente objetos de los que habitualmente no se conoce su tipo concreto, son los tipos genéricos.
- En su esencia, el término genéricos significa tipos parametrizados. Son importantes porque permiten crear clases, interfaces y métodos en los que el tipo de datos sobre los que operan se especifica como parámetro.
- Al definir una clase genérica, o bien una interfaz, en realidad estamos definiendo un meta-tipo/pseudotipo, una especie de plantilla a partir de la cual se crearán posteriormente clases/interfaces que actuarán como tipos concretos.

Se incorporan desde la JDK 5, extendiendo el sistema de tipos de datos, permitiendo que un método o tipo, opere con objetos de varios tipos, manteniendo seguridad en tiempo de ejecución.

- Son un forma de parametrizar “tipos” para declarar clases o interfaces.
- Solo funcionan con tipos de referencia, no soporta primitivos.
- Una clase o interfaz genérica puede admitir n cantidad de genéricos.
- Al igual un método genérico puede admitir n cantidad de genéricos.
- Se emplea el operador diamante o “< >”, para especificar el genérico.
- Generalmente se emplean letras mayúsculas para declararlos “<T>”, “< T, K >”
- Se puede delimitar los tipos a usar <T extends Number>

### **Tipos de dato genéricos (Generics):**

- Principales usos:
  - Colecciones
  - Interfaces

## Java Generics ejemplo

The diagram illustrates the use of Java Generics in a class named `MyGenerics`. The code is presented in a dark-themed editor with syntax highlighting. Four teal callout boxes with arrows point to specific parts of the code to explain its features:

- Puede contener atributos genéricos** (It can contain generic attributes) points to the `private T genericsfield;` declaration.
- Se indica que la clase contendrá genéricos** (It is indicated that the class will contain generics) points to the `<T>` type parameter in the class declaration.
- El constructor puede contener el genérico** (The constructor can contain the generic) points to the `T genericsfield` parameter in the constructor signature.
- Los métodos pueden recibir genéricos** (Methods can receive generics) points to the `T genericsfield` parameter in the `setGenericsfield` method signature.

```
public class MyGenerics <T>{  
    private T genericsfield;  
    private Integer field;  
  
    public MyGenerics(T genericsfield, Integer field) {  
        this.genericsfield = genericsfield;  
        this.field = field;  
    }  
    public T getGenericsfield() {  
        return genericsfield;  
    }  
    public void setGenericsfield(T genericsfield) {  
        this.genericsfield = genericsfield;  
    }  
}
```