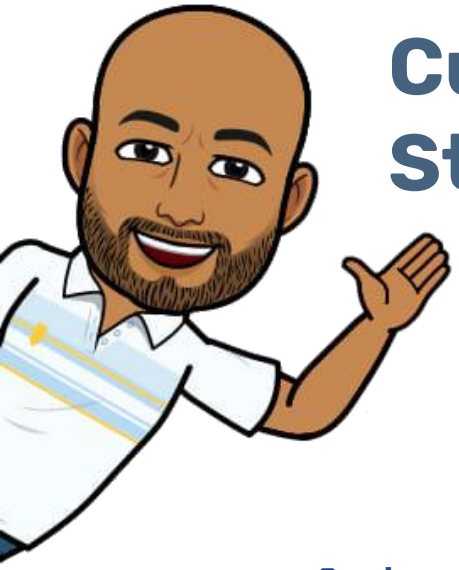

Curso de Java Standard

Ing. Octavio Robleto

octavio.robleto@gmail.com

https://octaviorobleto.com

Introducción

Java es un lenguaje de tipado fuerte lo que quiere decir que debemos asignar un tipo de dato a nuestros atributos y eso permite que en tiempo de compilación no se asigne un valor distinto al tipo de dato declarado.

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

En este ejemplo tenemos una clase que contiene dos atributos de tipo entero (código y numero); este tipo de objeto puede ser muy útil no solo para teléfono sino para otros casos, Documento, Domicilio, etc. Por lo que tendríamos que crear N cantidad de clases para cada caso de uso.

Pero que pasa si necesitamos un código que sea un **String** o un numero que sea un **double**?

Pudiéramos resolverlo declarando los tipos de datos como **Object** ya que por polimorfismo se convierte en cualquiera de sus hijos, incluyendo estructuras nuevas.

```java
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;
    }

}
```

Introducción

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

Recordemos **Object** puede convertirse en cualquiera de sus hijos, incluyendo todos los datos primitivos, lo que exigiría mas 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.

```java
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");
}
```

```java
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;
    }
}
```

Genéricos

Aquí es donde entran las clases genéricas o también llamados Tipo parametrizados incorporadas desde la versión 5 que nos 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 enviamos.

Al igual que al crear una clase como lo hemos realizado anteriormente colocamos lo siguiente:

1. Modificador de acceso.
2. Palabra reservada **class**.
3. Nombre de la clase.
4. Apertura llave Angular (Símbolo Menor Qué).
5. Letras genéricas en mayúsculas (si colocamos mas de una separar con comas).
6. Cierre llave Angular (Símbolo Mayor Qué).
7. Si lo necesitamos, las herencias e implementaciones de clases e interfaces.
8. Apertura y Cierre de llaves.

```java
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

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

Declaración del Objeto:

1. Identificador de la clase.
2. Apertura llave Angular (Símbolo Menor Qué).
3. Tipo de Objeto (si son mas de uno separar con comas). **No se permiten datos primitivos por lo que recurriremos a los objetos envoltorios.**
4. Cierre llave Angular (Símbolo Mayor Qué)
5. Identificador del objeto.

Instanciación del Objeto:

1. Identificador de la clase.
2. Apertura llave Angular (Símbolo Menor Qué).
3. Tipo de Objeto (si son mas de uno separar con comas).
4. Cierre llave Angular (Símbolo Mayor Qué).
5. Apertura de paréntesis (Constructor)
6. Argumentos del Constructor (si los requiere).
7. Cierre del paréntesis.

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

Desde el JDK 7 se permite utilizar el Operador Diamante "<>"; que es: no repetir el tipo de dato del genérico en la instanciación.

Ventajas

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

```
GenericaDosAtributos<Integer, Integer> telefono = new GenericaDosAtributos<>();

telefono.setCodigo("CABA");
telefono.setNumero(50124478);
```

Es importante aclarar que no es obligatorio enviar los tipos de datos al declarar un objeto de la clase genérica, lo que hará es asumir que los datos serán de tipo Object, aunque no se recomienda omitirlo.