



Programación Básica II - 2022

JAVA RECORD

Chamorro, Sebastian,
España Rocío,
Fernandez Candela,
Gasparrino Agostina,
Oviedo Joaquín.





Qué es?

- Es un nuevo tipo de clase en Java.
- Podemos usar el Records para escribir la misma estructura y obtener el mismo resultado, ahorrando el instanciar el constructor, los getter y setter de la clase

Características



- Son inmutables.
- Solo requiere tipo y nombre de los datos.
- No permiten atributos de instancia, pero si estáticos.
- No permiten herencia pero si interfaces.
- Se puede tener varios constructores.
- Se puede tener métodos adicionales
- No es recomendable usar para persistencia de base de datos

Vamos a implementar una
clase típica de DTO:

```
private final boolean status;

public ProductClass(int idProduct, String name, double price, boolean status) {
    this.idProduct = idProduct;
    this.name = name;
    this.price = price;
    this.status = status;
}

public int getIdProduct() {
    return idProduct;
}

public String getName() {
    return name;
}

public double getPrice() {
    return price;
}

public boolean isStatus() {
    return status;
}

@Override
public String toString() {
    return "ProductClass{" +
        "idProduct=" + idProduct +
        ", name='" + name + '\'' +
        ", price=" + price +
        ", status=" + status +
        '}';
}
```

Y en aquellos atributos instanciados en RECORD podríamos inicializarlos de la siguiente manera en nuestro main:

```
package com.mitocode.app;

import com.mitocode.records.ProductRecord;

public class App {

    public static void main(String[] args) {
        ProductRecord pr1 = new ProductRecord( idProduct: 1, name: "PS5", price: 999.99, status: true);
        System.out.println(pr1);
    }
}
```





Devolviendo por consola:

```
App
"C:\Program Files\Eclipse Adoptium\jdk-19.0.0.36-hotspot\bin\java.exe
ProductRecord[idProduct=1, name=PS5, price=999.99, status=true]

Process finished with exit code 0
```

Como podemos ver, no habría necesidad de sobrescribir un toString al utilizar RECORD.

Ahora en el caso que no queramos que todos los atributos sean inmutables, sólo algunos pocos, se realizaría de la siguiente manera:

```
package com.mitocode.records;

public record ProductRecord(int idProduct,
                            String name,
                            double price,
                            boolean status) {

    ..

    public ProductRecord(String name){
        this( idProduct: 0, name, price: 599.99, status: true);
    }
}
```

```
public ProductRecord(String name, double price){
    this( idProduct: 0, name, price, status: true);
}
```

Inicializando solo los valores que dejamos inmutables, porque lo demás en teoría variarían:

```
public class App {  
  
    public static void main(String[] args) {  
        ProductRecord pr1 = new ProductRecord( idProduct: 1, name: "PS5", price: 999.99, status: true);  
        ProductRecord pr2 = new ProductRecord( name: "PS4");  
        ProductRecord pr3 = new ProductRecord( name: "PS4", price: 799.99);  
  
        System.out.println(pr1);  
        System.out.println(pr2);  
        System.out.println(pr3);  
    }  
}
```

```
"C:\Program Files\Eclipse Adoptium\jdk-19.0.0.36-hotspot\bin\jav  
ProductRecord[idProduct=1, name=PS5, price=999.99, status=true]  
ProductRecord[idProduct=0, name=PS4, price=599.99, status=true]  
ProductRecord[idProduct=0, name=PS4, price=799.99, status=true]  
  
Process finished with exit code 0
```


También podríamos implementar métodos a nuestro Record:

```
public String nameLowerCase(){  
    return name.toLowerCase();  
}
```

Como transformar
nuestro name en
minúscula.

Y como no utilizamos métodos de
Getters y Setters, en nuestro main
llamamos solamente al atributo:

```
//System.out.println(pr2);  
System.out.println(pr3.name());  
System.out.println(pr3.nameLowerCase());  
}
```

Y dentro de esto, también
podríamos implementar métodos
estáticos:

```
//static  
public static void printYearOfProduct(){  
    System.out.println("2022");  
}
```

En donde, en nuestro main, lo llamaremos
únicamente por la clase Product Record:

```
//System.out.println(pr3.name());  
//System.out.println(pr3.nameLowerCa  
ProductRecord.printYearOfProduct();
```

Creando una interfaz

- Con un constructor público y abstracto por definición. Y luego agregando implements al ProductRecordconsola:

```
package com.mitocode.records;  
  
public interface ProductMachine {  
    void printCountryMachine();  
}
```

- Sobrescribiendo aquel método en esta clase:

```
public record ProductRecord(int idProduct,  
    1 usage  
    String name,  
    double price,  
    boolean status) implements ProductMachine{
```



Creando una interfaz

- Y este método se sobrescribiera en nuestra clase records para que este pueda utilizarla

```
@Override  
public void printCountryMachine() {  
    System.out.println("PERU");  
}
```



Y luego, otro Record donde agregaremos Category entre sus atributos:

```
package com.mitocode.records;


public record SuperProduct(int idProduct, String name, Category category) {
}
```

Por lo que en nuestro main, se podrá llamar de la siguiente manera:

```
SuperProduct sp = new SuperProduct( idProduct: 1, name: "RTX3080TI", new Category( idCategory: 1, name: "VIDEO-CARDS"));
System.out.println(sp);
```

Obteniendo la información propia:

```
"C:\Program Files\Eclipse Adoptium\jdk-19.0.0.36-hotspot\bin\java.exe" "-javaagent:D:\Program
SuperProduct[idProduct=1, name=RTX3080TI, category=Category[idCategory=1, name=VIDEO-CARDS]]
```

- 
- Si quisiéramos instanciar nuevos atributos, únicamente estos deberán ser estáticos e inmutables:

```
public static final String DEFAULT_PRODUCT = "COMPUTER";
```

- Y llamamos el atributo de la siguiente manera en el main:

```
//System.out.println(pr3);  
System.out.println(ProductRecord.DEFAULT_PRODUCT);
```