
# 1. Constructores estáticos

| | |
|---|---|
| Date | Empty |
| Status | Not started |
| Type | Empty |
| Units | 7. Uso avanzado de clases |

## Constructores estáticos

Como se vio en clases anteriores, un constructor es el método especial que me permite crear un objeto para poder utilizar cualquier método de la clase. Para poder crear un constructor corriente las restricciones era:

- tener un ámbito público
- no tener modificador de acceso

```
public class UnaClase { public UnaClase() { } public void unM
etodo(){ System.out.println("Ejecución de un método"); } }
```

De esta forma se crea un constructor de una clase y desde cualquier clase podrá ser utilizado para crear un objeto de la clase y utilizar sus métodos

```
public class Entrada { public static void main(String[] args)
{ UnaClase unaClase = new UnaClase(); unaClase.unMetodo(); }
}
```

Además de esta posibilidad, también existe la posibilidad de utilizar un método estático para poder crear un objeto. Si recordáis el modificador static indicaba que aquella variable o método donde fuese aplicado podría ser llamado de forma directa tan solo utilizando el nombre de la clase. Si esto lo aplicamos al ejemplo anterior y creamos un método estático, este podrá ser llamado directamente sin necesidad de tener un objeto de la clase

```
public class UnaClase { public UnaClase() { } public static v
oid unMétodoEstático(){ System.out.println("Un método estátic
o"); } }
```

```
public class Entrada { public static void main(String[] args)
{ UnaClase.unMétodoEstático(); } }
```

Con esta característica, se puede utilizar el método estático para crear y deber un objeto de forma directa, sin necesidad de utilizar la palabra reservada new

```
public class UnaClase { public UnaClase() { } public static U
naClase unMétodoEstático(){ UnaClase objeto = new UnaClase();
return objeto; } }
```

De forma que este método pueda ser llamado para generar un objeto

```
public class Entrada { public static void main(String[] args)
{ UnaClase objeto = UnaClase.unMétodoEstático(); } }
```

Esta capacidad nos permite desde una clase generar un objeto y acceder a sus métodos de forma directa. Por ejemplo imaginad una

clase que tiene una colección de datos que nos interesa que sean accedido (por ejemplo una lista de equipos de fútbol)

```
public class Equipo { private String nombre, pais; private in
t ranking; public Equipo(String nombre, String pais, int rank
ing) { this.nombre = nombre; this.pais = pais; this.ranking =
ranking; } public String getNombre() { return nombre; } publi
c String getPais() { return pais; } public int getRanking() {
return ranking; } }
```

```
public class DataSet { public ArrayList<Equipo> getEquiposEsp
ña(){ ArrayList<Equipo> equipos = new ArrayList<>(); equipo
s.add(new Equipo("FC Barcelona",1)); equipos.add(new Equipo("
Reeal Madrid",2)); equipos.add(new Equipo("Sevilla",3)); equi
pos.add(new Equipo("Real Sociedad",4)); equipos.add(new Equip
o("Getafe",5)); equipos.add(new Equipo("Real Sociedad",6)); r
eturn equipos; } public ArrayList<Equipo> getEquposItalia(){
ArrayList<Equipo> equipos = new ArrayList<>(); equipos.add(ne
w Equipo("Juventus",1)); equipos.add(new Equipo("Lazio",2));
equipos.add(new Equipo("Inter de Milán",3)); equipos.add(new
Equipo("Atalanta",4)); equipos.add(new Equipo("Roma",5)); equ
ipos.add(new Equipo("Nápoles",6)); return equipos } }
```

Si en la clase DataSet se crea un método estático que genere un objeto, se podrá acceder directamente a los métodos sin necesidad de tener el objeto creado, tan solo accediendo a los métodos que nos interesa

```
public class DataSet { public static DataSet newInstance() {
DataSet dataSet = new DataSet(); return dataSet; } public Arr
ayList<Equipo> getEquiposEspaña(){ ArrayList<Equipo> equipos
= new ArrayList<>(); equipos.add(new Equipo("FC Barcelona",
1)); equipos.add(new Equipo("Reeal Madrid",2)); equipos.add(n
ew Equipo("Sevilla",3)); equipos.add(new Equipo("Real Socieda
d",4)); equipos.add(new Equipo("Getafe",5)); equipos.add(new
Equipo("Real Sociedad",6)); return equipos; } public ArrayLis
t<Equipo> getEquposItalia(){ ArrayList<Equipo> equipos = new
ArrayList<>(); equipos.add(new Equipo("Juventus",1)); equipo
s.add(new Equipo("Lazio",2)); equipos.add(new Equipo("Inter d
e Milán",3)); equipos.add(new Equipo("Atalanta",4)); equipos.
add(new Equipo("Roma",5)); equipos.add(new Equipo("Nápoles",
```

```
6)); return equipos } }
```

El método newInstance podrá ser llamado de forma directa desde cualquier clase

```
public class Entrada { public static void main(String[] args)
{ ArrayList<Equipo> listaEspaña = DataSet.newInstance().getEq
uiposEspaña(); } }
```

En este ejemplo se llama al método newInstance el cual devuelve un objeto de tipo DataSet, y al mismo tiempo se llama al método getEquiposEspaña que devuelve un arraylist, pudiendo igualarlo a una variable del mismo tipo para utilizarla como sea necesario en la clase Entrada.

```
public class Entrada { public static void main(String[] args)
{ System.out.println("CLASIFICACIÓN ESPAÑA"); System.out.prin
tln("***********"); ArrayList<Equipo> listaEspaña = DataSet.n
ewInstance().getEquiposEspaña(); for (Equipo equipo: listaEsp
aña) { System.out.println(String.format("Posición %d %s",equi
po.getRanking(),equipo.getNombre())); } System.out.println
("\\nCLASIFICACIÓN ITALIA"); System.out.println("**********
*"); ArrayList<Equipo> listaItalia = DataSet.newInstance().ge
tEquposItalia(); for (Equipo equipo: listaItalia) { System.ou
t.println(String.format("Posición %d %s",equipo.getRanking(),
equipo.getNombre())); } } }
```