

# Herencia a través de un ejemplo

Corina Flores Villarroel

# Definición

- *La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.*



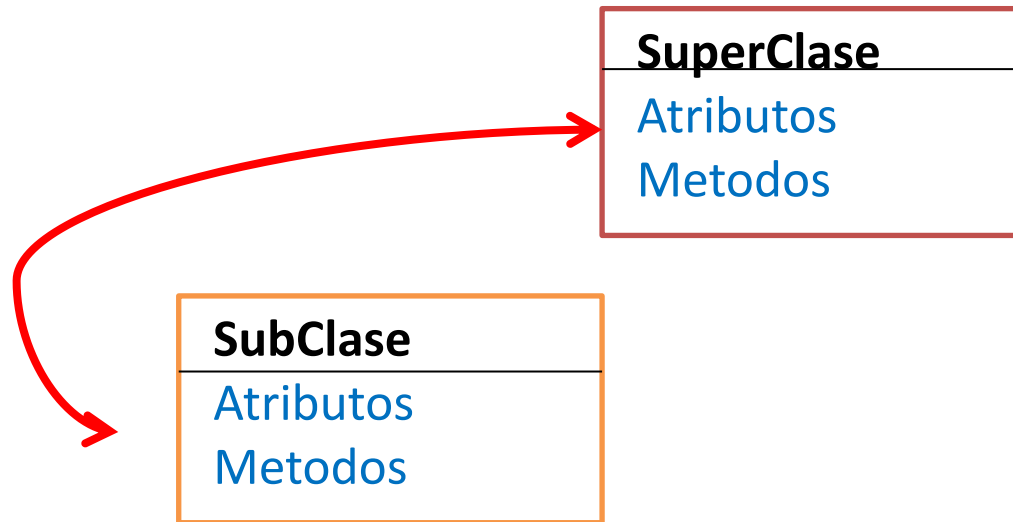
Nueva Clase

The diagram illustrates inheritance. It consists of two rectangular boxes. The top box is light red with a dark red border and contains the text 'Nueva Clase'. The bottom box is white with an orange border and contains the text 'Clase ya existente'. A vertical line connects the bottom of the top box to the top of the bottom box, representing the inheritance relationship.

Clase ya existente

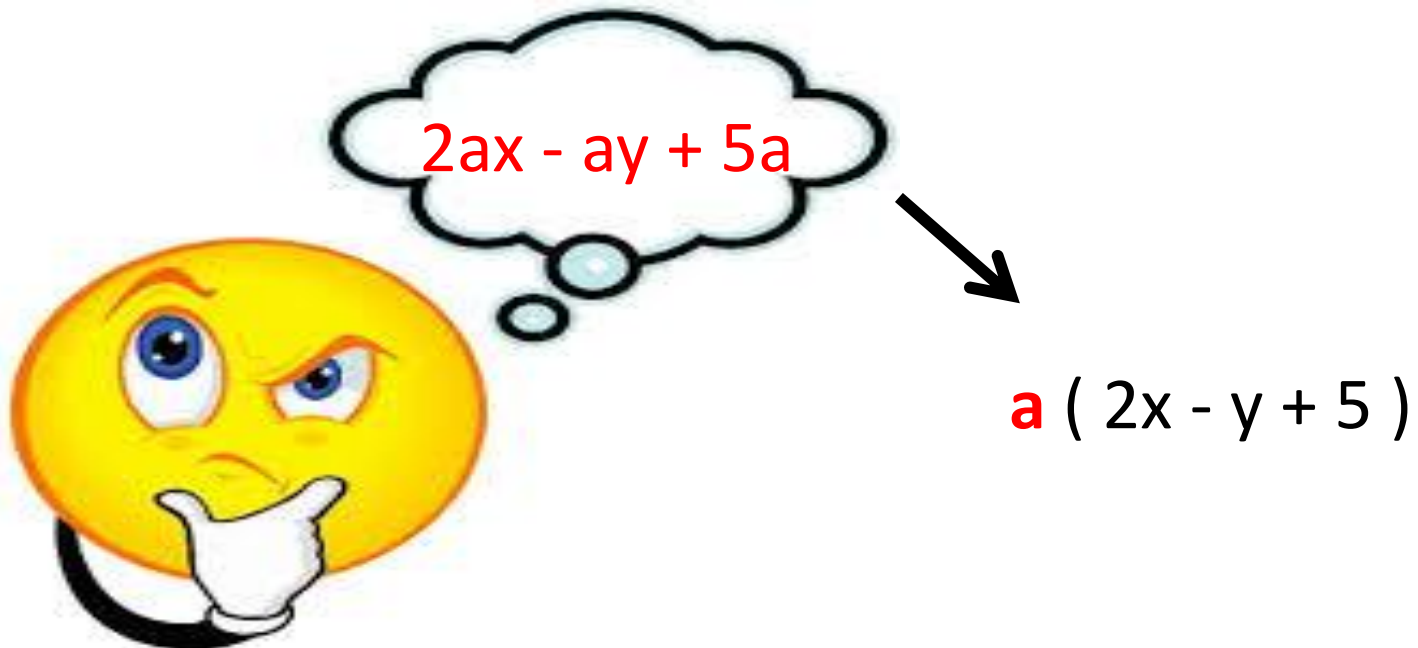
# ¿Qué permite ...?

- *La herencia permite compartir (automáticamente) **atributos y métodos** entre la super clase y subclases.*



# ¿Cómo se expresa ...?

- Herencia no es más que un “***Copy-Paste Dinámico***” o una forma de “***sacar factor común***” al código que escribimos.



# Ejemplo

- Vamos a simular el comportamiento que tendrían los diferentes integrantes de la selección boliviana de futbol; tanto los Futbolistas como el cuerpo técnico (Entrenadores, Masajistas, Utileros, Preparador Físico, etc...).

# Selección de futbol

- Futbolistas



# Selección de futbol



Entrenador



Uterero

- Cuerpo técnico



Preparador Físico



Masajista

# Ejemplo

- Para simular este comportamiento vamos a definir **tres clases** que van a representar a **objetos Futbolista, Entrenador y Masajista**.





# Ejemplo



## Futbolista

carnetIdent  
nombre  
apellidos  
edad  
numero  
...

Concentrarse()  
Entrenar()  
Viajar()  
JugarPartido()  
...

## Entrenador

carnetIdent  
nombre  
apellidos  
edad  
codEntren  
...

Concentrarse()  
Viajar()  
DirigirPartido()  
...

## Masajista

carnetIdent  
nombre  
apellidos  
edad  
Experiencia  
...

Concentrarse()  
Viajar()  
Masajear()  
...

# Y la nueva clase seria

SeleccionFutbol

carnetIdent

nombre

apellidos

edad

numero

...

setCi()

getCi()

...

- Estas clases representadas en código quedaría así ...

# Clase SeleccionFutbol

- `public class SeleccionFutbol`
- `{`
- `protected int ci;`
- `protected String nombre;`
- `protected String apellidos;`
- `protected int edad;`
- `protected String pos;`
- `protected int nro;`
- `// constructor, getter y setter`
- `public int setCi () {`
- `...`
- `}`
- `public void getCi() {`
- `...`
- `}`
- `....`
- `}`

# Clase Futbolista

- `public class Futbolista`
- `{`
- `private int ci;`
- `private String nombre;`
- `private String apellidos;`
- `private int edad;`
- `private String pos;`
- `private int nro;`
- `// constructor, getter y setter`
- `public void Concentrarse() {`
- `...`
- `}`
- `public void Viajar() {`
- `...`
- `}`
- `public void jugarPartido() {`
- `...`
- `}`
- `public void entrenar() {`
- `...`
- `}`
- `}`

# Clase Entrenador

- `public class Entrenador`
- `{`
- `private int ci;`
- `private String nombre;`
- `private String apellidos;`
- `private int edad;`
- `private String codEntren;`
- `// constructor, getter y setter`
- `public void Concentrarse() {`
- `...`
- `}`
- `public void Viajar() {`
- `...`
- `}`
- `public void dirigirPartido() {`
- `...`
- `}`
- `}`

# Clase Masajista

- `public class Masajista`
- `{`
- `private int ci;`
- `private String nombre;`
- `private String apellidos;`
- `private int edad;`
- `private int experiencia;`
- `// constructor, getter y setter`
- `public void Concentrarse() {`
- `...`
- `}`
- `public void Viajar() {`
- `...`
- `}`
- `public void Masajear() {`
- `...`
- `}`
- `}`

# Observando el código ...

- ¿puedes ver si hay instrucciones que se **duplican o repiten** en las tres clases?
- ¿dónde?



- ¿Qué **atributos** son comunes en las tres clases?

# Ejemplo



Futbolista	Entrenador	Masajista
carnetIdent nombre apellidos edad	carnetIdent nombre apellidos edad	carnetIdent nombre apellidos edad
numero ...	codEntren ...	experiencia ...
Concentrarse() Entrenar() Viajar() JugarPartido()	Concentrarse() Viajar() DirigirPartido() ...	Concentrarse() Viajar() Masajear() ...

...

- ¿Qué **métodos** son comunes en las tres clases?

# Ejemplo



## Futbolista

carnetIdent  
nombre  
apellidos  
edad  
numero  
...

Concentrarse()  
Viajar()

Entrenar()  
JugarPartido()

...

## Entrenador

carnetIdent  
nombre  
apellidos  
edad  
codEntren  
...

Concentrarse()  
Viajar()

DirigirPartido()  
...

## Masajista

carnetIdent  
nombre  
apellidos  
edad  
experiencia  
...

Concentrarse()  
Viajar()

Masajear()  
...

# Ejemplo



Futbolista	Entrenador	Masajista
carnetIdent nombre apellidos edad	carnetIdent nombre apellidos edad	carnetIdent nombre apellidos edad
numero ...	codEntren ...	experiencia ...
Concentrarse() Viajar() Entrenar() JugarPartido()	Concentrarse() Viajar() DirigirPartido() ...	Concentrarse() Viajar() Masajear() ...

...

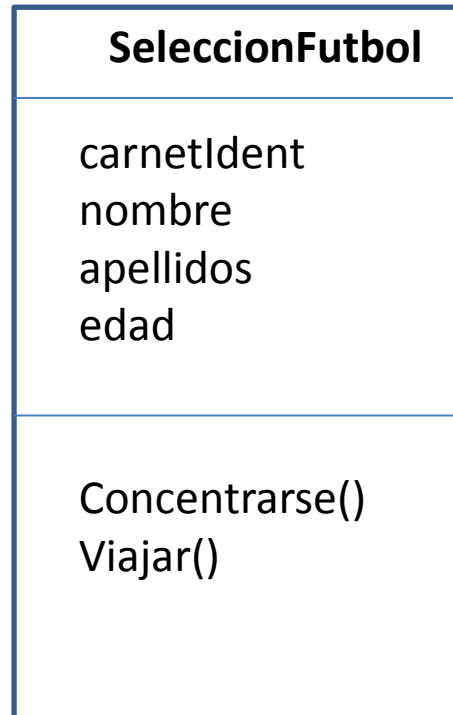
- Entonces, ¿qué hacemos?
- Sacar el factor común o lo que es lo mismo que **factorizar!**

- ¿Para qué?
- Para no escribir código demás ...

- ¿Cuál es la solución?
- Crear una nueva clase con el **código que es común** a las tres clases (a esta clase se le denominará en el concepto de herencia como **clase Padre o Superclase**)



# Clase Padre o Superclase



- ¿Y qué hacemos con el código que **no se repite** (atributos y métodos propios de cada clase o sea aquello que no es común)?

- Lo dejaremos en cada clase y denominaremos a éstas como **clases Hijas o Subclases** (estas clases heredarán de la clase padre **todos** los atributos y métodos públicos o protegidos y **no así los atributos y métodos privados**)

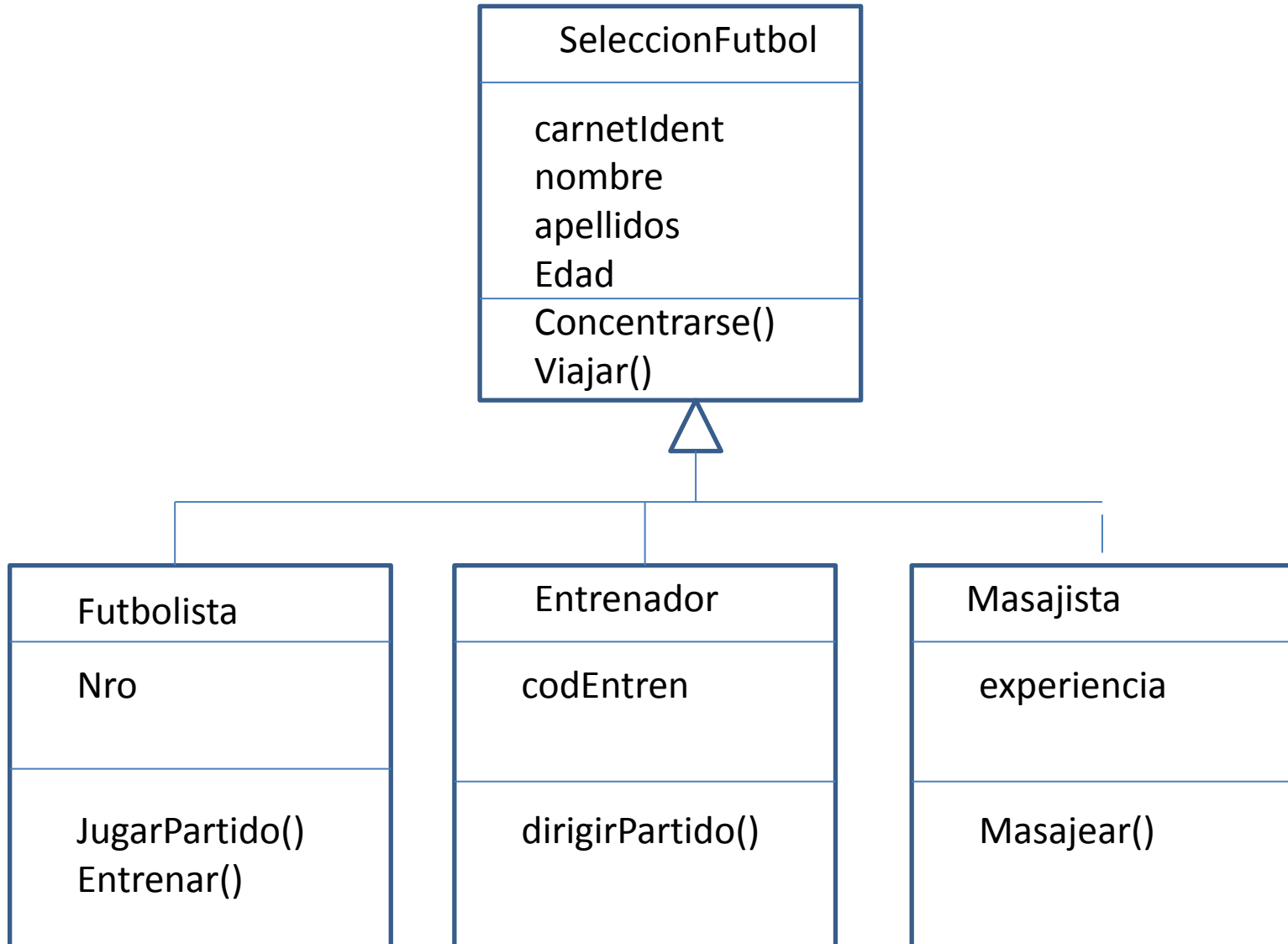
# Clases Hijas o Subclases

Futbolista
nro
JugarPartido() Entrenar()

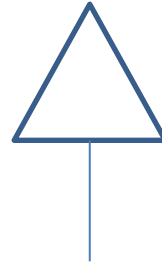
Entrenador
codEntren
dirigirPartido()

Masajista
experiencia
Masajear()

# Modelo de clases con herencia



# ¿Como representar la relación en el modelo?



Y se lee como **es un**

Ejemplo:

el **Futbolista** **es un** miembro de la **SeleccionFutbol**

# Clase SeleccionFutbol

- public class SeleccionFutbol
- {
- - protected int ci;
  - protected String nombre;
  - protected String apellidos;
  - protected int edad;
- // constructor, getter y setter
- public void concentrarse() {
- - ...
- }
- public void viajar() {
- - ...
- }
- }

# Las demás clases quedan así

```
class Futbolista extends SeleccionFutbol
{
    private String pos;
    private int nro;

    // constructor, getter y setter

    public void jugarPartido() {
        ...
    }

    public void entrenar() {
        ...
    }
}
```

```
class Masajista extends SeleccionFutbol
{
    private int experiencia;

    // constructor, getter y setter

    public void masajear() {
        ...
    }
}
```



# ¿Como representar la relación en el código?

Con la palabra reservada **extends**

Ejemplo:

```
class Futbolista extends SeleccionFutbol  
{
```

# Palabra reservada **extends**

- **extends**, indica a la clase hija cual va a ser su clase padre, por ejemplo en la clase Futbolista al poner “*public class Futbolista extends SeleccionFutbol*” le estamos indicando a la clase ‘Futbolista’ que su clase padre es la clase ‘SeleccionFutbol’

# Palabra reservada **extends**

- Por ejemplo: al poner **extends** estamos haciendo un “*copy-paste dinámico*” diciendo a la clase “**Futbolista**” que se ‘copie’ todos los atributos y métodos públicos o protegidos de la clase “**SeleccionFutbol**”

# Un nuevo modificador de acceso

protected

# ¿Cuándo usar protected?

- **protected**, sirve para indicar un tipo de visibilidad de los atributos y métodos de la clase padre.
- Solo se utiliza con herencia
- Se usa para indicar que cuando un atributo o método es 'protected' o protegido, **solo es visible ese atributo o método desde una de las clases hijas** y no desde otra clase.

# ¿Cómo usar protected?

Ejemplo:

Como atributos de la **superclase** (SeleccionFutbol) tendremos:

- carnetIdent
- nombre
- apellidos
- Edad

**declarados como protected.**

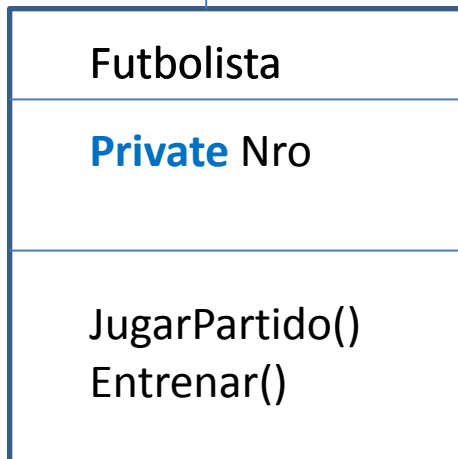
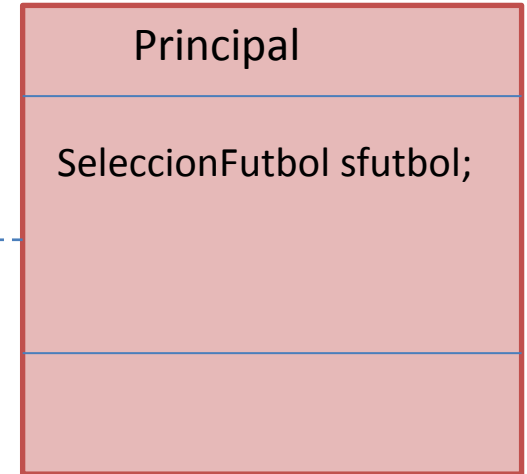
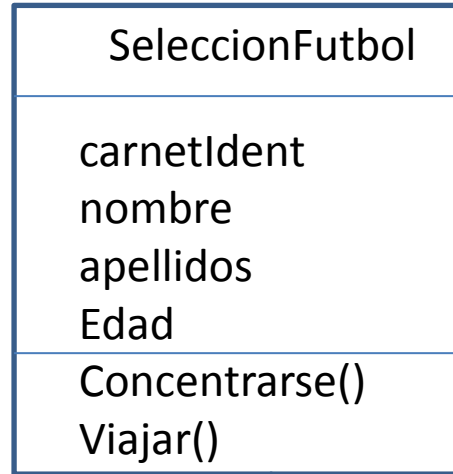
# ¿Cómo usar protected?

Ejemplo:

En la **subclase** (**Futbolista**) se trabajará con el campo adicional **numero** declarado como **private**.

# ¿Cómo usar protected?

PROTECTED





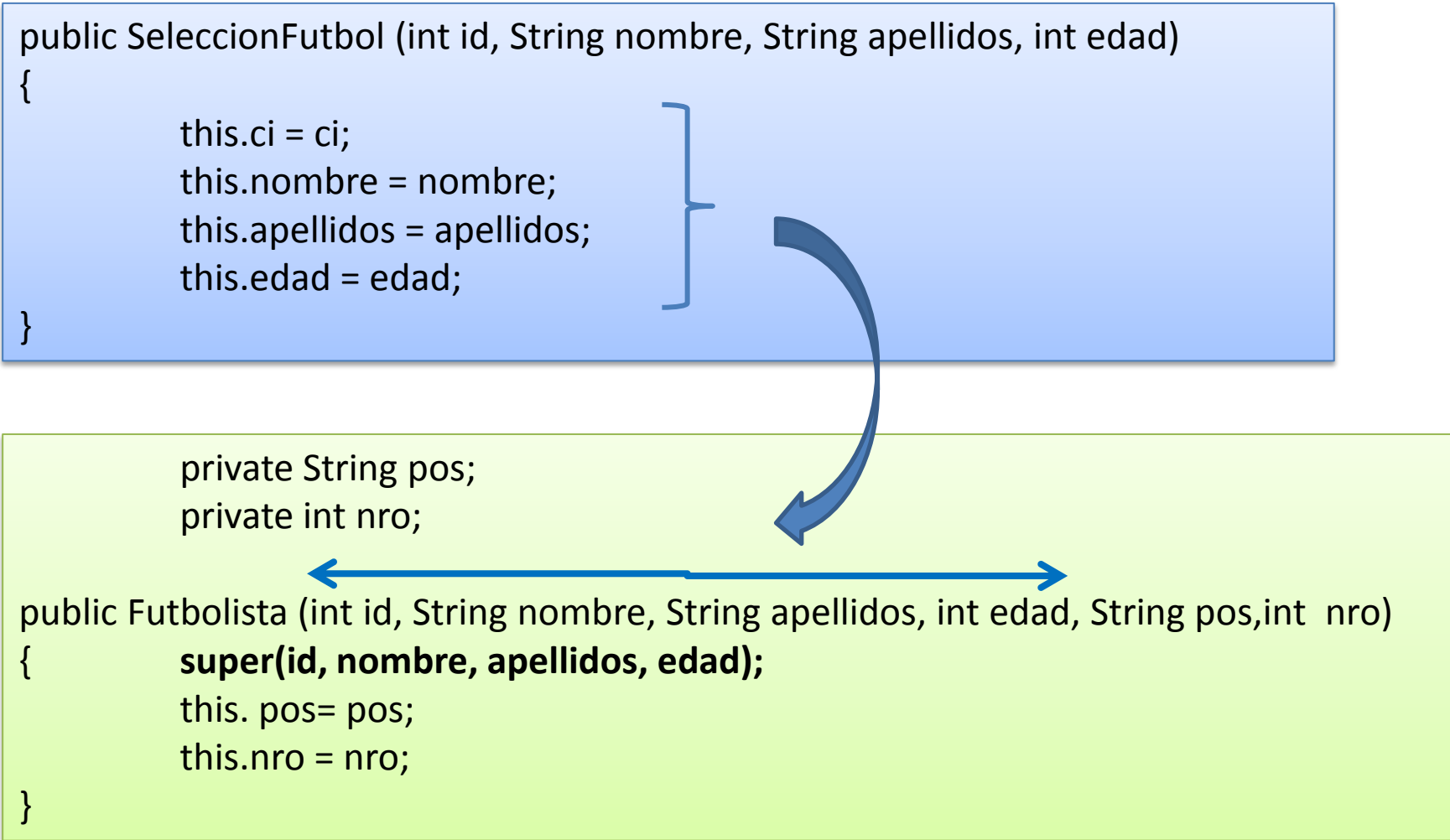
# ¿Cómo hacemos referencia a los atributos de la clase padre?

**super**

- **super**, sirve para llamar al constructor de la clase padre. Así por ejemplo:

# ¿Cómo se utiliza? (atributos)

```
public SeleccionFutbol (int id, String nombre, String apellidos, int edad)
{
    this.ci = ci;
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.edad = edad;
}
```



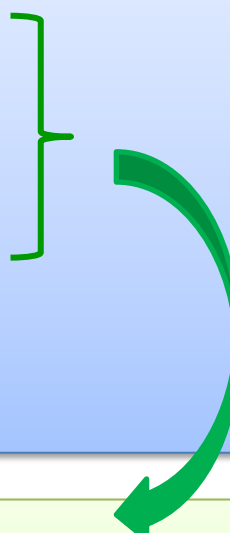
```
private String pos;
private int nro;
```

```
public Futbolista (int id, String nombre, String apellidos, int edad, String pos,int  nro)
{
    super(id, nombre, apellidos, edad);
    this. pos= pos;
    this.nro = nro;
}
```

# ¿Cómo se utiliza? (métodos)

```
Class SeleccionFutbol
{ ...
    public String toString () {
        return "Informacion:\n" +
            "\tCI: "+ci+"\n" +
            "\tNombre: "+nombre+"\n" +
            "\tApellido: "+apellidos+"\n" +
            "\tEdad: "+edad+" anhos\n";
    }
}
```

```
Class Futbolista
{ ...
    public String toString () {
        return super.toString()+ "\tPosicion: "+pos+"\n" +
            "\tNumero: "+nro+"\n";
    }
}
```



# Importante!

- `super` es la primera instrucción que se escribe en el cuerpo del constructor

# ¿Qué hemos ganado?

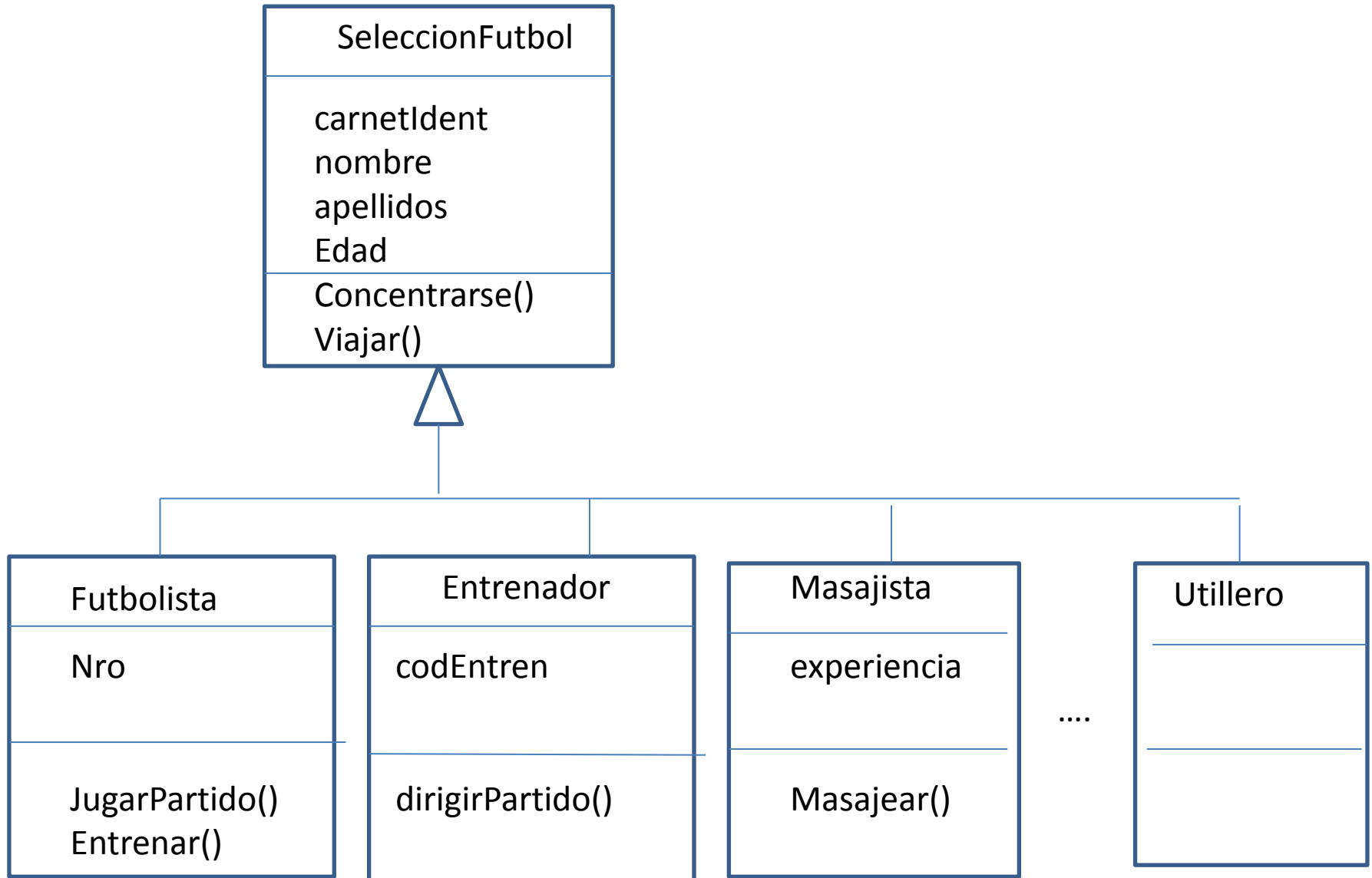
- Un código mucho más ***limpio, estructurado y con menos líneas de código***, lo que lo hace más ***legible***, por tanto ***un código reutilizable***

# ¿Qué pasa si quisiéramos añadir más clases?

- Por ejemplo una clase Médico, Utilero, Preparador físico, etc. que pertenezcan también al equipo técnico de la selección boliviana.

- ... podemos hacer de forma muy sencilla ya que en la clase padre (SeleccionFutbol) tenemos implementado parte de sus datos y de su comportamiento y solo habrá que implementar los atributos y métodos propios de esa clase. Es aquí donde se puede apreciar la utilidad de la herencia!

# Modelo de clases con herencia





# Tarea

- La entrega de la tarea es para el día 30 de diciembre
- Entregar por correo electrónico a la dirección: [corina.flores1@gmail.com](mailto:corina.flores1@gmail.com) hasta las 23:50 del día indicado.

# Tarea

- Partiendo del ejemplo EjercicioHerencia3 publicado en [www.cs.umss.edu.bo](http://www.cs.umss.edu.bo) se pide:
  - Mostrar la selección de futbol con los 11 futbolistas, un entrenador y un masajista.
  - Mostrar solo a los futbolistas que deben concentrarse.

# Tarea

- Mostrar la lista de los futbolistas que deben viajar.
- Mostrar al entrenador que va a dirigir el partido de futbol
- Mostrar la lista de jugadores que deben jugar el partido ordenados por número
- Mostrar los jugadores que juegan en la posición de delanteros

# Tarea

- Mostrar el nombre y apellido del masajista y sus años de experiencia.
- Dado el apellido de un futbolista, buscar si se encuentra en la lista de los futbolistas convocados a entrenar.