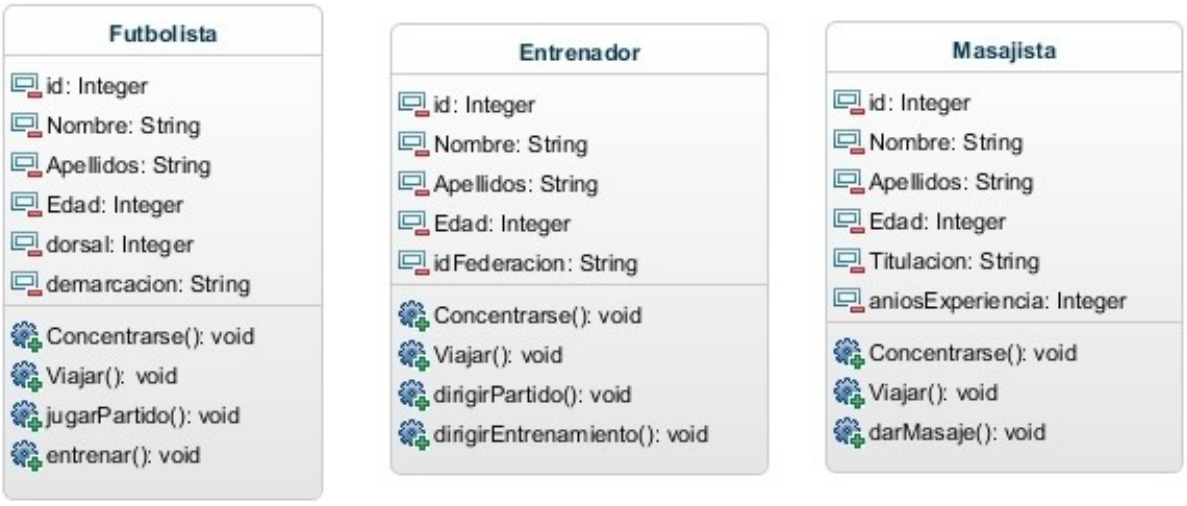
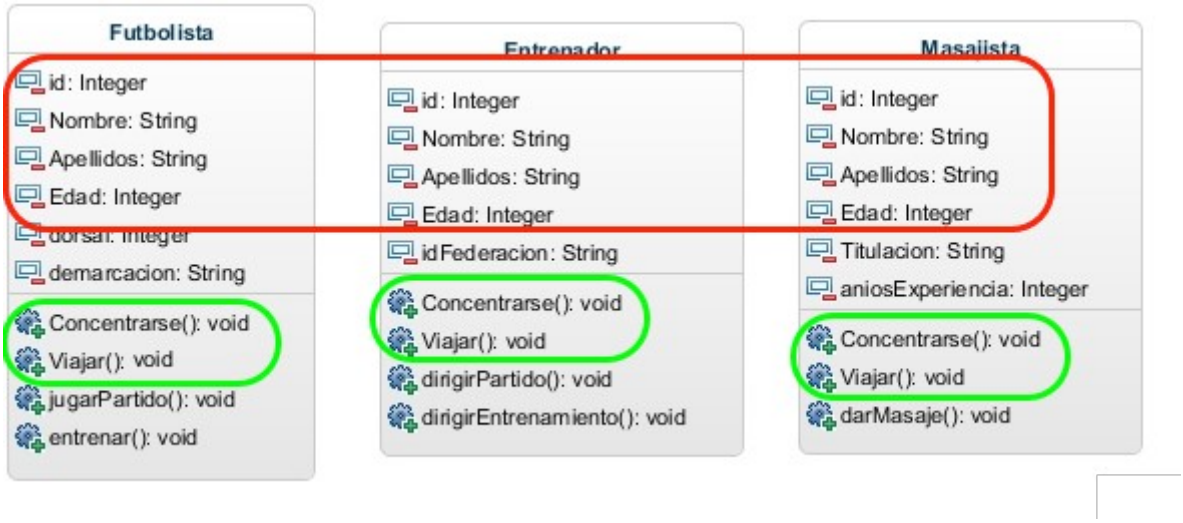
"La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente. La herencia permite compartir automáticamente métodos y datos entre clases, subclases y objetos.". Así de primeras esta definición es un poco difícil de digerir para aquellos que estéis empezando con la POO, así que vamos a intentar digerir esta definición con un ejemplo en el que veremos que la herencia no es más que un "Copy-Paste Dinámico" o una forma de "sacar factor común" al código que escribimos.

El ejemplo que proponemos es un caso en el que vamos a simular el comportamiento que tendrían los diferentes integrantes de la selección española de futbol; tanto los Futbolistas como el cuerpo técnico (Entrenadores, Masajistas, etc…). Para simular este comportamiento vamos a definir tres clases que van a representaran a objetos Futbolista, Entrenador y Masajista. De cada unos de ellos vamos a necesitar algunos datos que reflejaremos en los atributos y una serie de acciones que reflejaremos en sus métodos. Estos atributos y métodos los mostramos en el siguiente diagrama de clases:



Como se puede observar, vemos que en las tres clases tenemos atributos y métodos que son iguales ya que los tres tienen los atributos *id, Nombre, Apellidos y Edad*; y los tres tienen los métodos de *Viajar* y *Concentrarse:*



A nivel de código tenemos lo siguiente tras ver el diagrama de clases:

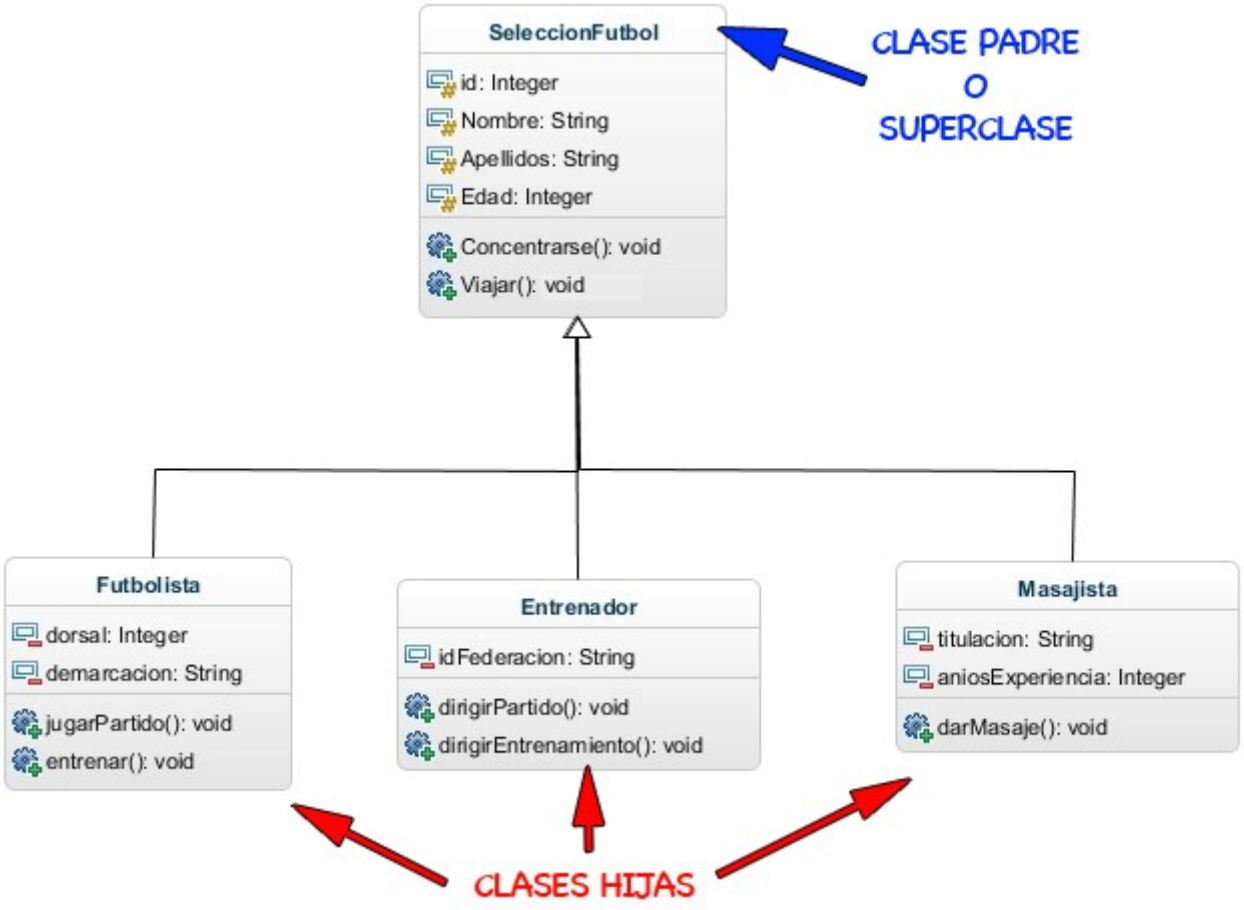
```java
public class Futbolista
{

        private int id;
        private String Nombre;
        private String Apellidos;
        private int Edad;
        private int dorsal;
        private String demarcacion;

        // constructor, getter y setter

        public void Concentrarse() {
                ...
        }

        public void Viajar() {
                ...
        }

        public void jugarPartido() {
                ...
        }

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

```java
public class Entrenador
{

        private int id;
        private String Nombre;
        private String Apellidos;
        private int Edad;
        private String idFederacion;

        // constructor, getter y setter

        public void Concentrarse() {
                ...
        }

        public void Viajar() {
                ...
        }

        public void dirigirPartido() {
                ...
        }

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

```java
public class Masajista
{

        private int id;
        private String Nombre;
        private String Apellidos;
        private int Edad;
        private String Titulacion;
        private int aniosExperiencia;

        // constructor, getter y setter

        public void Concentrarse() {
                ...
        }

        public void Viajar() {
                ...
        }

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

Lo que podemos ver en este punto es que estamos escribiendo mucho código repetido ya que las tres clases tienen métodos y atributos comunes, de ahi y como veremos enseguida, decimos que la herencia consiste en "sacar factor común" para no escribir código de más, por tanto lo que haremos sera crearnos una clase con el "código que es común a las tres clases" (a esta clase se le denomina en la herencia como "Clase Padre o SuperClase") y el código que es especifico de cada clase, lo dejaremos en ella, siendo denominadas estas clases como "Clases Hijas", **las cuales heredan de la clase padre todos los atributos y métodos públicos o protegidos.**

Es muy importante decir que las clases hijas no van a heredar nunca los atributos y métodos privados de la clase padre.
En resumen, al "sacar factor común" y aplicar herencia, tenemos las siguientes clases.



A nivel de código, las clases quedarían implementadas de la siguiente forma:

```java
public class SeleccionFutbol
{

        protected int id;
        protected String Nombre;
        protected String Apellidos;
        protected int Edad;

        // constructor, getter y setter

        public void Concentrarse() {
                ...
        }

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

```java
public class Futbolista extends SeleccionFutbol
{
        private int dorsal;
        private String demarcacion;

        public Futbolista() {
                super();
        }

        // getter y setter

        public void jugarPartido() {
                ...
        }

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

```java
public class Entrenador extends SeleccionFutbol
{

        private String idFederacion;

        public Entrenador() {
                super();
        }

        // getter y setter

        public void dirigirPartido() {
                ...
        }

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

```java
public class Masajista extends SeleccionFutbol
{

        private String Titulacion;
        private int aniosExperiencia;

        public Masajista() {
                super();
        }

        // getter y setter

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

Como podéis observar ahora queda un código mucho más limpio, estructurado y con menos líneas de código, lo que lo hace más legible, cosa que es muy importante y lo que todavía lo hace más importante es que es un código reutilizable, lo que significa que ahora si queremos añadir más clases a nuestra aplicación como por ejemplo una clase Médico, Utiller@, Jefe/a de prensa etc. que pertenezcan también al equipo técnico de la selección Española, lo 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. ¿Empezáis a ver la utilidad de la herencia?.

Habréis podido observar dos palabras reservadas "nuevas" como son "***extends***", "***protected***" y "super". Pues bien, ahora vamos a explicar el significado de ellas:

extends: Esta palabra reservada, indica a la clase hija cual va a ser su clase padre, es decir que 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' o dicho de otra manera para que se entienda mejor, al poner esto 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'. De aquí viene esa 'definición' que dimos de que la herencia en un 'copy-paste dinámico'.

protected: sirve para indicar un tipo de visibilidad de los atributos y métodos de la clase padre y significa que cuando un atributo es 'protected' o protegido, solo es visible ese atributo o método desde una de las clases hijas y no desde otra clase.

super: sirve para llamar al constructor de la clase padre.

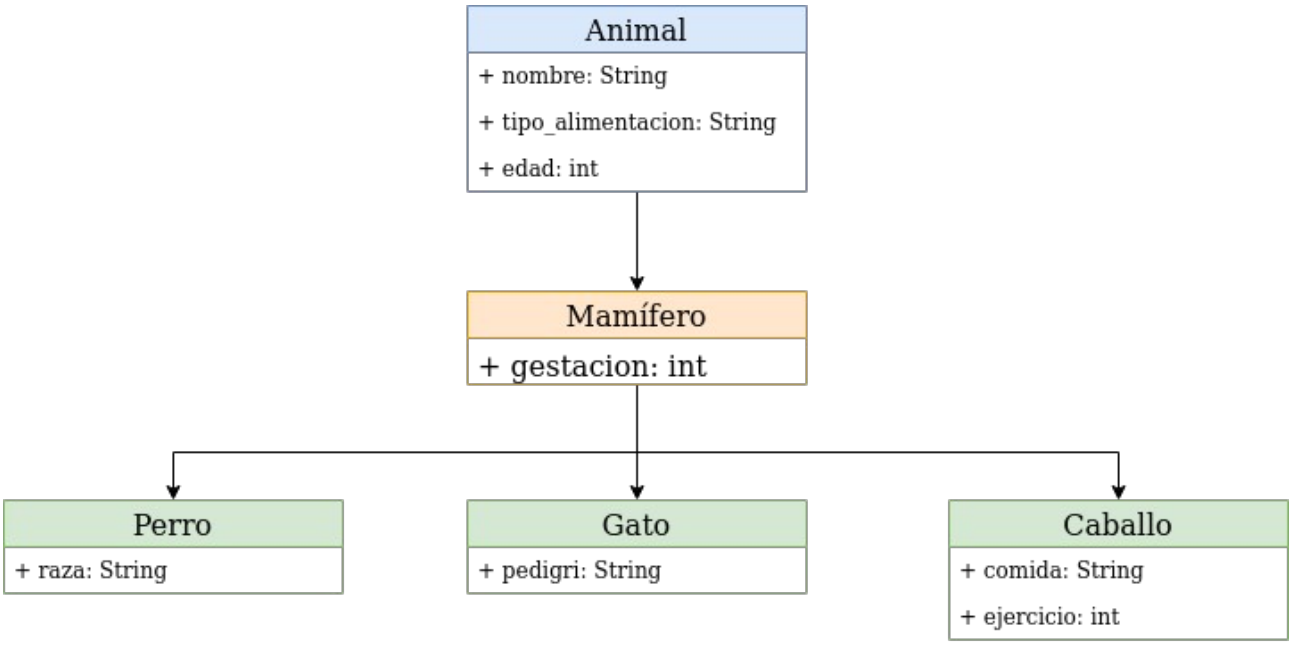
```
public class SeleccionFutbol {

        ......

        public SeleccionFutbol() {
        }

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

```java
public class Futbolista extends SeleccionFutbol {
        ......
        public Futbolista() {
                super();
        }

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

Para ver este funcionamiento de forma clara y sencilla vamos a trabajar con un objeto de cada clase y vamos a ver como se crean y de que forma ejecutan sus método. Para ello empecemos mostrando el siguiente fragmento de código:

```java
public class Main {

        // ArrayList de objetos SeleccionFutbol. Idenpendientemente de la clase hija a la que pertenezca el objeto
        public static ArrayList<SeleccionFutbol> integrantes = new ArrayList<SeleccionFutbol>();

        public static void main(String[] args) {

                Entrenador delBosque = new Entrenador(1, "Vicente", "Del Bosque", 60, "284EZ89");
                Futbolista iniesta = new Futbolista(2, "Andres", "Iniesta", 29, 6, "Interior Derecho");
                Masajista raulMartinez = new Masajista(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);

                integrantes.add(delBosque);
                integrantes.add(iniesta);
                integrantes.add(raulMartinez);

                // CONCENTRACION
                System.out.println("Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)");
                for (SeleccionFutbol integrante : integrantes) {
                        System.out.print(integrante.getNombre()+" "+integrante.getApellidos()+" -> ");
                        integrante.Concentrarse();
                }

                // VIAJE
                System.out.println("nTodos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)");
                for (SeleccionFutbol integrante : integrantes) {
                        System.out.print(integrante.getNombre()+" "+integrante.getApellidos()+" -> ");
                        integrante.Viajar();
                }
        ......
```

EJEMPLO



Tenemos nuestra clase Animal, con los atributos:
•nombre, tipo de alimentación Y edad

La clase hija Mamífero, con el atributo:

•gestación (periodo de gestación en meses)

Y 3 clases hijas de Mamífero:

Perro, con el atributo. raza

Gato, con el atributo: pedigrí

Caballo, con los atributos: comida Y ejercicio (veces/día que hace ejercicio)

Cada una de las clases hijas tiene diferentes atributos, pero comparten las mismas propiedades de un Animal.

```java
public class Animal {
    private String nombre;
    private String tipo_alimentacion;
    private int edad;

    public Animal() {

    }
    public Animal(String nombre, String tipo_alimentacion, int edad){
        this.nombre = nombre;
        this.tipo_alimentacion = tipo_alimentacion;
        this.edad = edad;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getTipo_alimentacion() {
        return tipo_alimentacion;
    }
    public void setTipo_alimentacion(String tipo_alimentacion) {
        this.tipo_alimentacion = tipo_alimentacion;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public String toString(){

        return this.nombre + ", " + this.tipo_alimentacion + ", " + this.edad;
    }
}
```

```java
public class Mamifero extends Animal{
    private int gestacion;

    public Mamifero() {
        super();
    }
    public Mamifero(String nombre, String tipo_alimentacion, int edad, int gestacion) {
        super(nombre, tipo_alimentacion, edad);
        this.gestacion = gestacion;
    }
    public int getGestacion(){

        return this.gestacion;
    }
    public void setGestacion(int gestacion){
        this.gestacion = gestacion;
    }
    public String toString() {

        return super.toString() + ", " + this.gestacion;
    }
}
```

```java
public class Perro extends Mamifero {
    private String raza;

    public Perro() {
        super();
    }
    public Perro(String nombre, String tipo_alimentacion, int edad, int gestacion, String raza) {
        super(nombre, tipo_alimentacion, edad, gestacion);
        this.raza = raza;
    }
    public String getRaza() {

        return this.raza;
    }
    public void setRaza(String raza) {
        this.raza = raza;
    }
    public String toString() {

        return super.toString() + ", " + this.raza;
    }
}
```

```java
public class Gato extends Mamifero {
    private String pedigri;

    public Gato() {
        super();
    }
    public Gato(String nombre, String tipo_alimentacion, int edad, int gestacion, String pedigri) {
        super(nombre, tipo_alimentacion, edad, gestacion);
        this.pedigri = pedigri;
    }
    public String getPedigri() {

        return this.pedigri;
    }
    public void setPedigri(String pedigri) {
        this.pedigri = pedigri;
    }
    public String toString() {

        return super.toString() + ", " + this.pedigri;
    }
}
```

```java
public class Caballo extends Mamifero {
    private String comida;
    private int ejercicio = 1; //nº de veces/día que hace ejercicio

    public Caballo() {
        super();
    }
    public Caballo(String nombre, String tipo_alimentacion, int edad, int gestacion, String comida, int ejercicio) {
        super(nombre, tipo_alimentacion, edad, gestacion);
        this.comida = comida;
        this.ejercicio = ejercicio;
    }

    public String getComida() {
        return comida;
    }

    public void setComida(String comida) {
        this.comida = comida;
    }

    public int getEjercicio() {
        return ejercicio;
    }

    public void setEjercicio(int ejercicio) {
        this.ejercicio = ejercicio;
    }
    public String toString() {

        return super.toString() + ", " + this.comida + ", " + this.ejercicio;
    }
}
```

```java
public class EjemploAnimal {
    public static void main(String[] args) {
        Animal animal = new Animal("Gustavo", "insectos", 3);
        Mamifero mamifero = new Mamifero("Moby Dick", "plancton", 5, 12);
        Perro perro = new Perro("Tobby", "carne", 7, 2, "caniche");
        Gato gato = new Gato("Tom", "pescado", 4, 4, "siamés");
        Caballo caballo = new Caballo("Rocinante", "hierba", 8, 12, "español", 2);

        System.out.println(animal);
        System.out.println(mamifero);
        System.out.println(perro);
        System.out.println(gato);
        System.out.println(caballo);
        caballo.setNombre("Rucio");
        caballo.setEjercicio(3);
        System.out.println(caballo);
    }
}
```