Herència Simple

L'Herència és un dels 4 pilars bàsics de la programació orientada a objectes (POO) juntament amb l'Abstracció, Encapsulació i Polimorfisme.

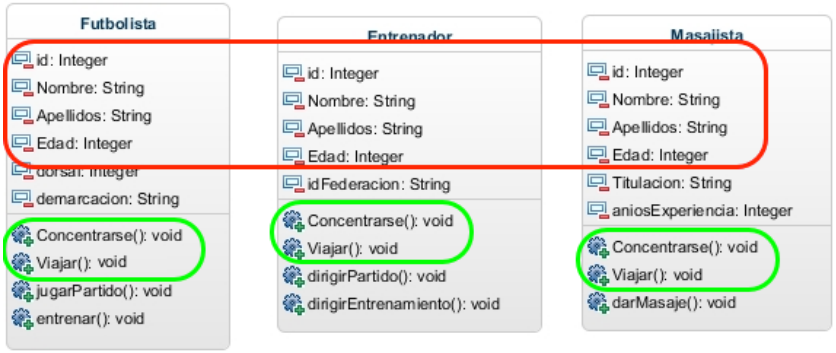
L'herència és un mecanisme que permet la definició d'una classe a partir de la definició d'una altra ja existent, de manera que la classe nova hereta tots els seus atributs i mètodes de la classe. L'herència permet compartir automàticament mètodes i dades entre classes, subclasses i objectes.

A la classe ja existent se la denomina **superclasse, classe base o classe pare**. A la nova classe se la denomina **subclasse, classe derivada o classe filla**. La subclasse és una **EXTENSIÓ** de la superclasse

Per entendre esta definició, podem considerar l'herència com un **«Copy-Paste Dinàmic»** o una manera de treure **«Factor Comú»** al codi que escrivim. La subclasse és una EXTENSIÓ de la superclasse

Exemple: Imaginem que volem simular el comportament que tindrien els diferents integrants de usa selecció nacional de futbol (Futbolistes, Entrenadors i Massatgistes).

En les tres classes trobem atributs i mètodes que són exactament iguals:



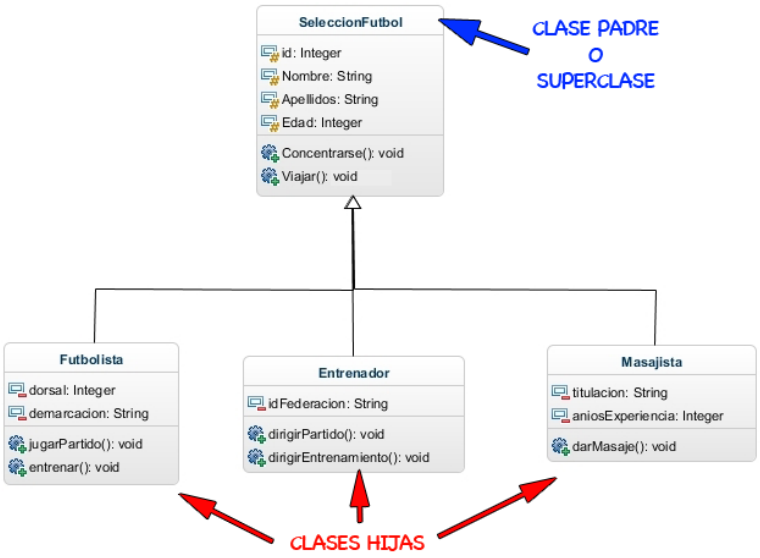
En codi (sense Constructors, Getters i Setters per abreviar) tindríem:

```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 añosExperiencia;

// constructor, getter y setter
  public void Concentrarse()  { … }

  public void Viajar()  { … }

  public void darMasaje()  { … }

}
```

Podem observar que les 3 classes tenen molts atributs i mètodes comuns, i per tant estem escrivint molt de **codi repetit**, per tant, el que farem es crear una **nova classe** amb el codi que és comú a les tres classes, es a dir, traiem **"factor comú"** del codi repetit.



En Herència, a esta nova classe la denominem "Classe Pare o SuperClasse, i tindrà tots els atributs i els mètodes comuns

Les classes derivades es denominen "Classes Filles o SubClasses".

El codi que és específic de cada SubClasse, el deixarem en elles.

Cada subclasse **hereta de la classe pare tots els atributs i mètodes públics o protegits.**

Les Subclasses NO poden heretar atributs o mètodes PRIVATE, sols PUBLIC O PROTECTED

VISIBILITAT DELS MEMBRES D'UNA CLASSE

Modificador	Classe	SubClasse	Paquet	Tots
public	SI	SI	SI	SI
private	SI	NO	NO	NO
protected	SI	SI	SI	NO
No declarat (sense modificador)	SI	SI	NO	NO

SUPERCLASSE SeleccionFutbol

```
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()  { … }

}
```

Subclasse Futbolista

```
public class Futbolista extends
SeleccionFutbol {

  private int dorsal;
  private String demarcacion;

public Futbolista() {
   super();
}

// getters y setters

  public void jugarPartido()  { … }

  public void entrenar()  { … }

}
```

Subclasse Entrenador

```
public class Entrenador extends
SeleccionFutbol {

  private String idFederacion;

  public Entrenador() {
   super();
}

// getters y setters

  public void dirigirPartido() { … }

  public void dirigirEntreno() { … }

}
```

Subclasse Masajista

```
public class Masajista extends
SeleccionFutbol {

  private String Titulacion;
  private int añosExperiencia;

  public Masajista() {
   super();
}

// getters y setters

  public void darMasaje()  { … }

}
```

Amb **l'herència** aconseguim un codi molt **més net**, **estructurat** i amb **menys línies de codi**, la qual cosa ho fa **més llegible**, i més important, és un codi **reutilitzable**.

Reutilitzable perquè:
si ara volem afegir més classes a la nostra aplicació com per exemple una classe Metge, Utiller@, Cap/a de premsa etc. que també formen part de l'equip tècnic de la selecció de futbol, el podem fer de forma molt senzilla ja que en la superclasse (SeleccionFutbol) tenim implementat part de les seues dades i del seu comportament i només caldrà implementar els atributs i mètodes propis d'aqueixa classe.

EXTENDS, PROTECTED, SUPER.

- **Extends**: Indica a la **subclasse** qui va a ser la seua **superclasse**, es a dir, de qui va a heretar mètodes i atributs.

public class Futbolista **extends** SeleccionFutbol

Amb esta instrucció estem indicant a la classe 'Futbolista' que la seua classe pare és la classe 'SeleccionFutbol' es a dir, estem fent un "**copy-paste dinàmic**" dient a la classe 'Futbolista' que es '**copie**' tots els atributs i mètodes **públics o protegits** de la classe 'SeleccionFutbol'.

- **Potected:** Indica un tipus de visibilitat dels atributs i mètodes de la classe pare i significa que quan un atribut és 'protected' o protegit, este atribut o mètode **només és visible** des d'una de les **classes filles** i no des d'una altra classe.

- **Super:** Serveix per a cridar al constructor de la classe pare.

```
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;
	}
	......
```

```
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;
	}
	......
```

El constructor de la subclasse **invoca al constructor de la superclasse**, per això utilitzem **OBLIGATÒRIAMENT** la paraula reservada **SUPER(-paràmetres del constructor-);** en la primera línia del Constructor.

¿Com treballem amb aquestes classes?. Per a veure aquest funcionament de manera clara i senzilla treballarem amb un objecte de cada classe i veurem com es creen i de que manera executen els seus mètode:

```
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 luisEnrique = new Entrenador(1, "Luis Enrique", "Martínez", 45, "284EZ89");
	  Futbolista ramos = new Futbolista(2, "Sergio", "Ramos", 35, 6, "Defensa");
	  Masajista raulMartinez = new Masajista(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);

	  integrantes.add(luisEnrique);
	  integrantes.add(ramos);
	  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();
	}

	// ENTRENAMIENTO
	System.out.println("nEntrenamiento: Solamente el entrenador y el futbolista tiene metodos para entrenar:");
	System.out.print(luisEnrique.getNombre()+" "+ luisEnrique.getApellidos()+" -> ");
	luisEnrique.dirigirEntrenamiento();
	System.out.print(ramos.getNombre()+" "+ramos.getApellidos()+" -> ");
		ramos.entrenar();

	// MASAJE
	System.out.println("nMasaje: Solo el masajista tiene el método para dar un masaje:");
	System.out.print(raulMartinez.getNombre()+" "+raulMartinez.getApellidos()+" -> ");
	raulMartinez.darMasaje();

	// PARTIDO DE FUTBOL
	System.out.println("nPartido de Fútbol: Solamente el entrenador y el futbolista tiene metodos para el partido de fútbol:");
	System.out.print(luisEnrique.getNombre()+" "+ luisEnrique.getApellidos()+" -> ");
	luisEnrique.dirigirPartido();
	System.out.print(ramos.getNombre()+" "+ramos.getApellidos()+" -> ");
	ramos.jugarPartido();
```

En este exemple, ens creem un objecte de cada classe, passant-li els atributs al constructor com a paràmetre i després "sorprenentment" els fiquem en un "ArrayList" d'objectes de la classe "SeleccionFutbol" que és la classe pare. Això evidentment t'ho permet fer ja que tots els objectes són fills de la mateixa classe pare. Després com veieu, recorrem el ArrayList i executem els seus mètodes "comuns" com són el 'Concentrar-se' i el 'Viatjar'.

La inicialització d'un objecte d'una subclasse compren 2 parts:

1.- **Invocació** al constructor de la superclasse ,amb la instrucció super();

2.- **Resta** de les **instruccions** del constructor de la subclasse.

Execució

```
Todos integrantes comienzan una concentracion. (Todos ejecutan el mismo método)
Luis Enrique -> Concentrarse
Sergio Ramos -> Concentrarse
Raúl Martínez -> Concentrarse

Todos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)
Luis Enrique -> Viajar
Sergio Ramos -> Viajar
Raúl Martínez -> Viajar
```

```
Entrenamiento: Solamente el entrenador y el futbolista tiene metodos para entrenar:
Luis Enrique -> Dirige un entrenamiento
Sergio Ramos -> Entrena

Masaje: Solo el masajista tiene el método para dar un masaje:
Raúl Martínez -> Da un masaje

Partido de Fútbol: Solamente el entrenador y el futbolista tiene métodos para el partido de
fútbol:
Luis Enrique -> Dirige un partido
Sergio Ramos -> Juega un partido
```