Polimorfisme (II). INTERFACES.

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

Per entendre el concepte d'Interfície (Interface) cal tindre clar els conceptes d'Herència i Classes Abstractes.

Una classe abstracta és una classe que no es pot instanciar (no es poden crear objectes d'eixa classe) però sí es pot definir atributs i implementar mètodes amb l'objectiu que les seues subclasses (per herència) els puguen utilitzar.

En POO una **classe abstracta** és una classe que no es pot instanciar (no es poden crear objectes d'eixa classe) però **sí** es pot **definir atributs i implementar mètodes** amb l'objectiu que les seues subclasses (per herència) els puguen utilitzar.

Una interficie és un pas mes enllà d'una classe abstracta, i la pode, definir com:

INTERFICIE: És una **classe abstracta** **PURA**, en la que **TOTS** els seus mètodes **són abstractes**, i per tant no es poden implementar en la classe interfície. utilitzar.

Si definíem una classe com una plantilla per a crear objectes, una interfície és una plantilla per a crear classes. Per tant:

Les Interfícies s'utilitzen per a establir la FORMA que ha de tindre una classe.

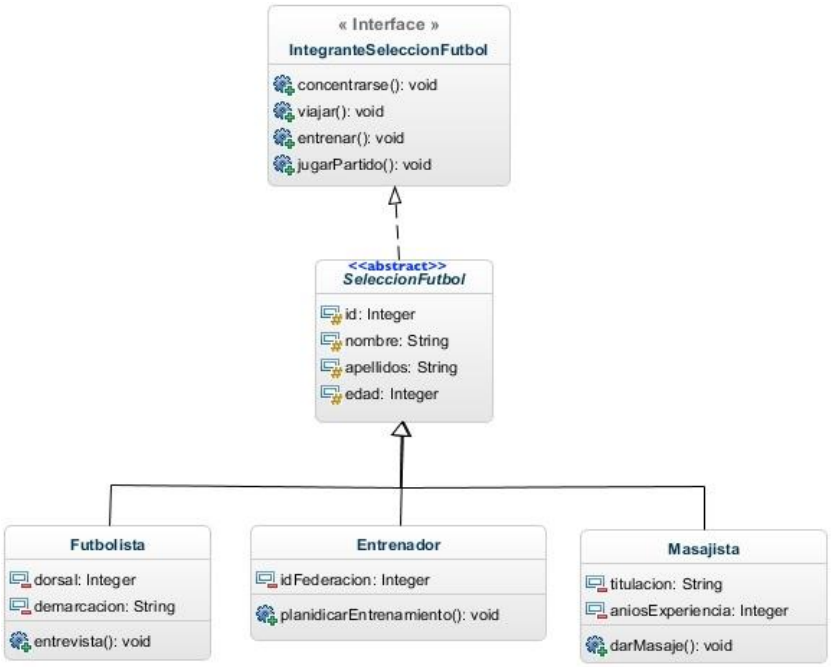
Un aspecte fonamental de les interfícies en Java és separar **l'especificació d'una classe** (ALLÒ QUE FA) de la **implementació** (COM HO FA) . Amb açò aconseguim programes més robustos i amb menys errors.

Cal tindre amb compte que:

- Una interfície **no es pot instanciar en objectes**, només serveix per a implementar classes.
- Una classe **pot implementar diverses interfícies** (separades per comes).
- Una classe que implementa una interfície ha de proporcionar **implementació per a tots i cadascun dels mètodes** definits en la interfície.
- Les classes que implementen una interfície que té **definides constants** poden usar-les en qualsevol part del codi de la classe, simplement indicant el seu nom.

Per poder entendre esta definició, veiem un exemple ja conegut del tema d'Herència: Integrants de la selecció de futbol.

Nota: En este exemple, no apareixen ni els constructors ni els mètodes **getter**'s i **sette**r's amb l'objectiu de fer simplificar l'exemple, encara que deurien aparèixer per tal de respectar el principi de **ENCAPSULAMENT** de la POO.



En aquest exemple trobem una classe **IntegranteSeleccionFutbol**, que és una **ITERFACE** (classe abstracta pura) on tenim quatre mètodes definits (els definim però **NO** els **IMPLEMENTEM**)

INTERFACE:

Al definir una **Interface** estem creant una **"plantilla de classes"**.

Totes les classes que implementen esta **Interface** hauran d'implementar **OBLIGATÒRIAMENT** tots els mètodes definits en la **Interface.**

Al ser una **CLASSE ABSTRACTA PURA**, no cal especificar els mètodes amb modificadors

```
public interface IntegranteSeleccionFutbol {

    void concentrarse();

    void viajar();

    void entrenar();

    void jugarPartido();
}
```

No posem abstract, ni públic ni private ni protected

La següent classe que trobem és **SeleccionFutbol** que implementa la Interfície **IntegranteSelecconFutbol,** amb la paraula **implements**.

Al implementar la interfície amb **implements**, estem obligats a implementar els mètodes **concentrarse**(), **viajar**(), **entrenar**() i **jugarPartido**().

```
public abstract class SeleccionFutbol implements IntegranteSeleccionFutbol {

    protected int id;

    protected String nombre;

    protected String apellidos;

    protected int edad;

    // Constructor, getter y setter


    public void concentrarse() {
        System.out.println("Concentrarse (Clase Padre)");
    }

    public void viajar() {
        System.out.println("Viajar (Clase Padre)");
    }

    public void entrenar() {
        System.out.println("Entrenar (Clase Padre)");
    }

    public void jugarPartido() {
        System.out.println("Asiste al Partido de Fútbol (Clase Padre)");
    }

}
```

IMPORTANT

Com la classe SeleccionFutbol l'hem declarat com una **CLASSE ABSTRACTA**, podríem **NO** definir estos mètodes i **obligar** a les subclasses **Futbolista, Entrenador i Masajista** a implementarlos.

En este exemple, s'implementen en la superclasse i es redefineixen en les subclasses.

El codi de les tres subclasses, aplicant herència i redefinint els mètodes de la superclasse quedaria:

Subclasse Futbolista

```
public class Futbolista extends SeleccionFutbol {

 private int dorsal;
 private String demarcacion;

    // Constructor, getter y setter

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;
}

 public int getDorsal() {
    return dorsal;
}

public void setDorsal(int dorsal) {
    this.dorsal = dorsal;
}

public String getDemarcacion() {
    return demarcacion;
}

public void setDemarcacion(String demarcacion)
{
    this.demarcacion = demarcacion;
}

            // Métodes

@Override
public void entrenar() {
    System.out.println("Realiza un entrenamiento (Clase Futbolista)");
}

@Override
public void jugarPartido() {
    System.out.println("Juega un Partido (Clase Futbolista)");
}

public void entrevista() {
    System.out.println("Da una Entrevista");
}
}
```

Subclasse Entrenador

```
public class Entrenador extends SeleccionFutbol
{
  private int idFederacion;

    // Constructor, getter y setter

public Entrenador(int id, String nombre, String apellidos, int edad, int idFederacion) {

    super(id, nombre, apellidos, edad);
    this.setIdFederacion(idFederacion);
 }

  public int getIdFederacion() {
     return idFederacion;
 }

   public void setIdFederacion(int idFederacion) {
     this.idFederacion = idFederacion;
  }

            // Métodes

@Override
public void entrenar() {
    System.out.println("Dirige un entrenamiento (Clase Entrenador)");

}

@Override
 public void jugarPartido() {
    System.out.println("Dirige un Partido (Clase Entrenador)");
}

public void planificarEntrenamiento() {
    System.out.println("Planificar un Entrenamiento");
}

}
```

Subclasse Masajista

```
public class Masajista extends SeleccionFutbol {

private String titulacion;
private int aniosExperiencia;

    // Constructor, getter y setter

public Masajista(int id, String nombre, String apellidos, int edad, String titulacion, int aniosExperiencia)
{
    super(id, nombre, apellidos, edad);
    this.titulacion = titulacion;
    this.aniosExperiencia = aniosExperiencia;
}

public String getTitulacion() {
    return titulacion;
}

public void setTitulacion(String titulacion) {
    this.titulacion = titulacion;
}

public int getAniosExperiencia() {
    return aniosExperiencia;
}

public void setAniosExperiencia(int aniosExperiencia)
{
    this.aniosExperiencia = aniosExperiencia;
}

            // Métodes

@Override
public void entrenar() {
    System.out.println("Da asistencia en el entrenamiento (Clase Masajista)");
}

public void darMasaje() {
    System.out.println("Da un Masaje");
}
}
```

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.

El programa principal i l'eixida per pantalla són iguals als de l'exemple utilitzats en l'apartat d'Herència,

El codi es mostra a continuació:

```java
public class PruebaSeleccion {

    // 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) {

        SeleccionFutbol luisEnrique = new Entrenador(1, "Luis Enrique", "Martinez", 48, 28489);
        SeleccionFutbol pedri = new Futbolista(2, "Pedro", "González", 18, 8, "Mediapunta");
        SeleccionFutbol raulMartinez = new Masajista(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);
        integrantes.add(luisEnrique);
        integrantes.add(pedri);
        integrantes.add(raulMartinez);

        // CONCENTRACION
        System.out.println("Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)\n");
        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: Todos los integrantes tienen su función en un entrenamiento (Especialización)");
        for (SeleccionFutbol integrante : integrantes) {
            System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");
            integrante.entrenar();
        }

        // PARTIDO DE FUTBOL
        System.out.println("\nPartido de Fútbol: Todos los integrantes tienen su función en un partido (Especialización)");
        for (SeleccionFutbol integrante : integrantes) {
            System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");
            integrante.jugarPartido();
        }

        // PLANIFICAR ENTRENAMIENTO
        System.out.println("\nPlanificar Entrenamiento: Solo el entrenador tiene el método para planificar un entrenamiento:");
        System.out.print(luisEnrique.getNombre() + " " + luisEnrique.getApellidos() + " -> ");
        ((Entrenador) luisEnrique).planificarEntrenamiento();

        // ENTREVISTA
        System.out.println("\nEntrevista: Solo el futbolista tiene el método para dar una entrevista:");
        System.out.print(pedri.getNombre() + " " + pedri.getApellidos() + " -> ");
        ((Futbolista) pedri).entrevista();

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

EIXIDA PER PANTALLA:

```
Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)

Luis Enrique Martinez -> Concentrarse (Clase Padre)
Pedro González -> Concentrarse (Clase Padre)
Raúl Martinez -> Concentrarse (Clase Padre)

Todos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)
Luis Enrique Martinez -> Viajar (Clase Padre)
Pedro González -> Viajar (Clase Padre)
Raúl Martinez -> Viajar (Clase Padre)

Entrenamiento: Todos los integrantes tienen su función en un entrenamiento (Especialización)
Luis Enrique Martinez -> Dirige un entrenamiento (Clase Entrenador)
Pedro González -> Realiza un entrenamiento (Clase Futbolista)
Raúl Martinez -> Da asistencia en el entrenamiento (Clase Masajista)

Partido de Fútbol: Todos los integrantes tienen su función en un partido (Especialización)
Luis Enrique Martinez -> Dirige un Partido (Clase Entrenador)
Pedro González -> Juega un Partido (Clase Futbolista)
Raúl Martinez -> Asiste al Partido de Fútbol (Clase Padre)

Planificar Entrenamiento: Solo el entrenador tiene el método para planificar un entrenamiento:
Luis Enrique Martinez -> Planificar un Entrenamiento

Entrevista: Solo el futbolista tiene el método para dar una entrevista:
Pedro González -> Da una Entrevista

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